Introduction to Deep Learning

Lecture 01

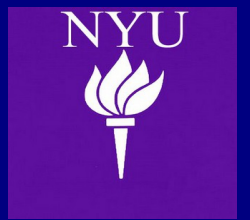
Yann Le Cun

Facebook AI Research,

Center for Data Science, NYU

Courant Institute of Mathematical Sciences, NYU

http://yann.lecun.com



Deep Learning = Learning Representations/Features

Y LeCun
MA Ranzato

- **The traditional model of pattern recognition (since the late 50's)**
  - Fixed/engineered features (or fixed kernel) + trainable classifier

hand-crafted Feature Extractor → "Simple" Trainable Classifier →

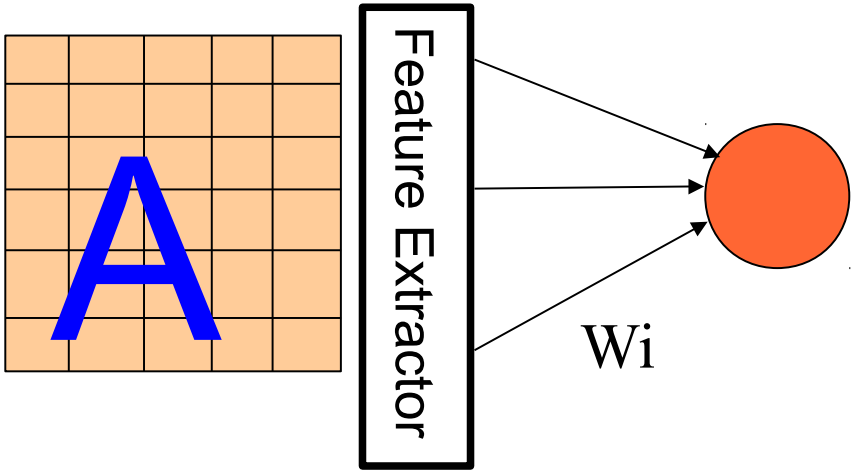
- **End-to-end learning / Feature learning / Deep learning**
  - Trainable features (or kernel) + trainable classifier

Trainable Feature Extractor → Trainable Classifier →

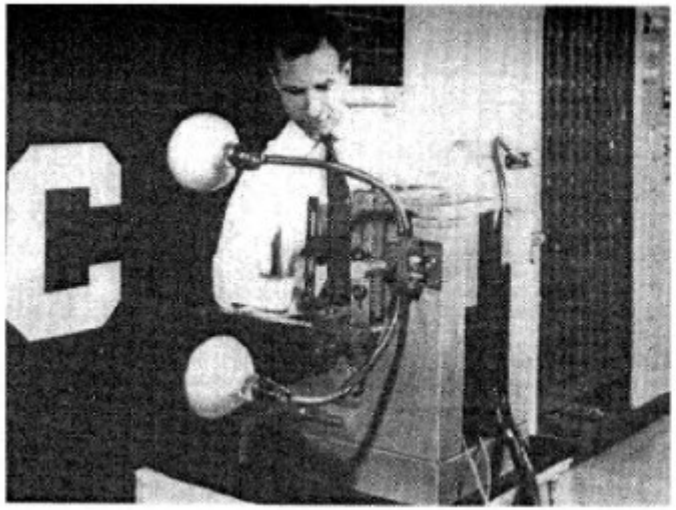
This Basic Model has not evolved much since the 50's

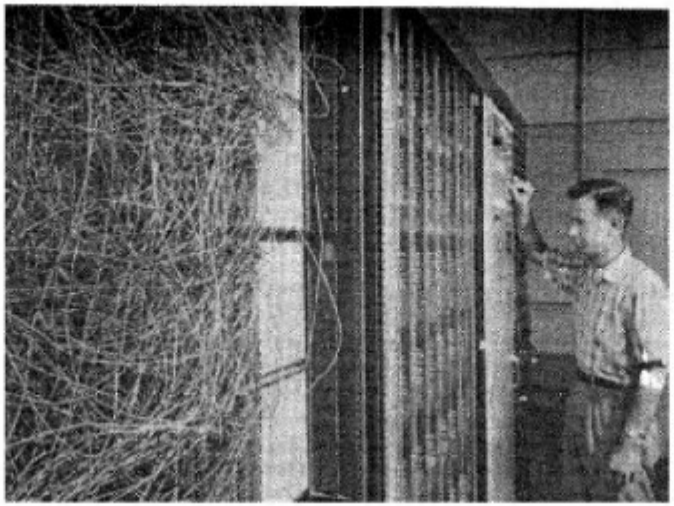
Y LeCun
MA Ranzato

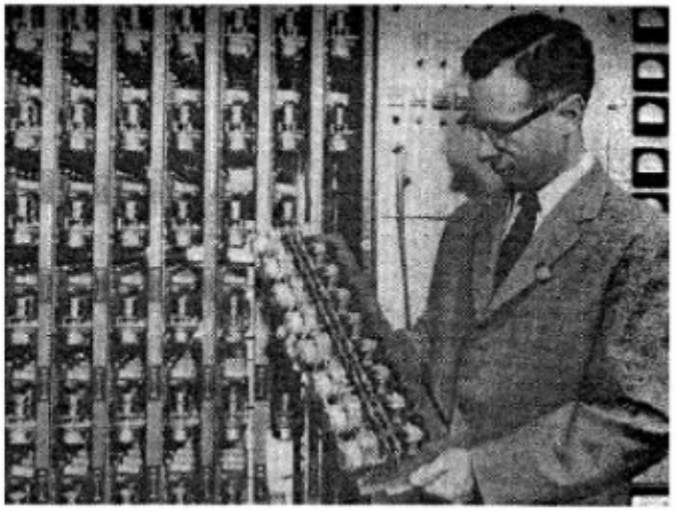
- **The first learning machine: the Perceptron**
  - Built at Cornell in 1960
- **The Perceptron was a linear classifier on top of a simple feature extractor**
- **The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.**



$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$

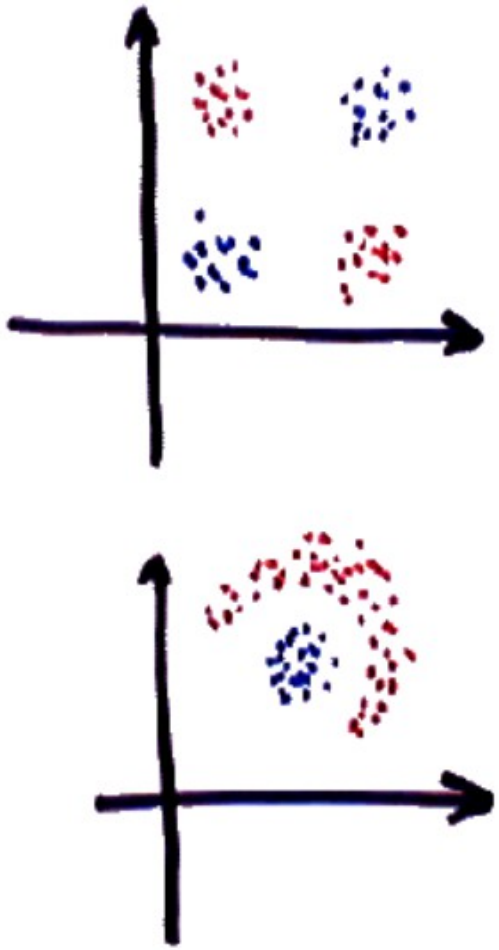







Linear Machines
And their limitations

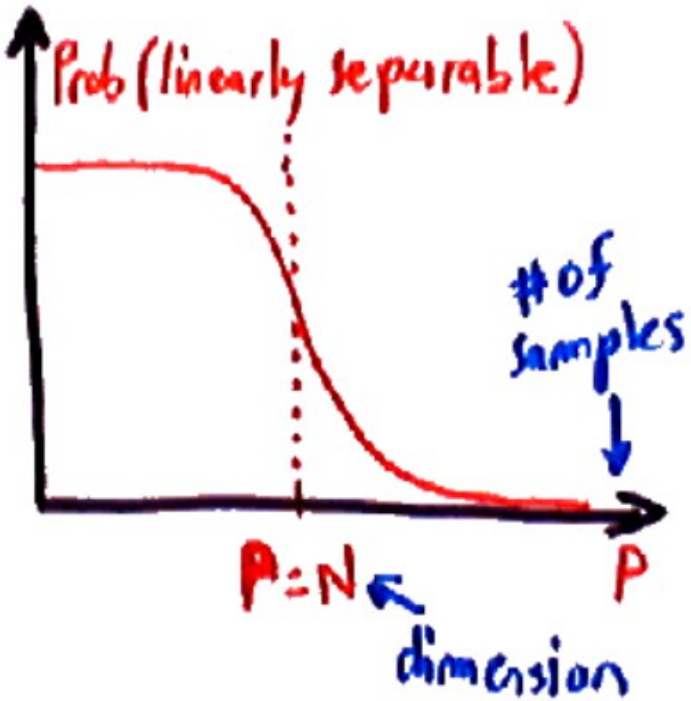
Limitations of Linear Machines



The *Linearly separable* dichotomies are the partitions that are realizable by a linear classifier (the boundary between the classes is a hyperplane).

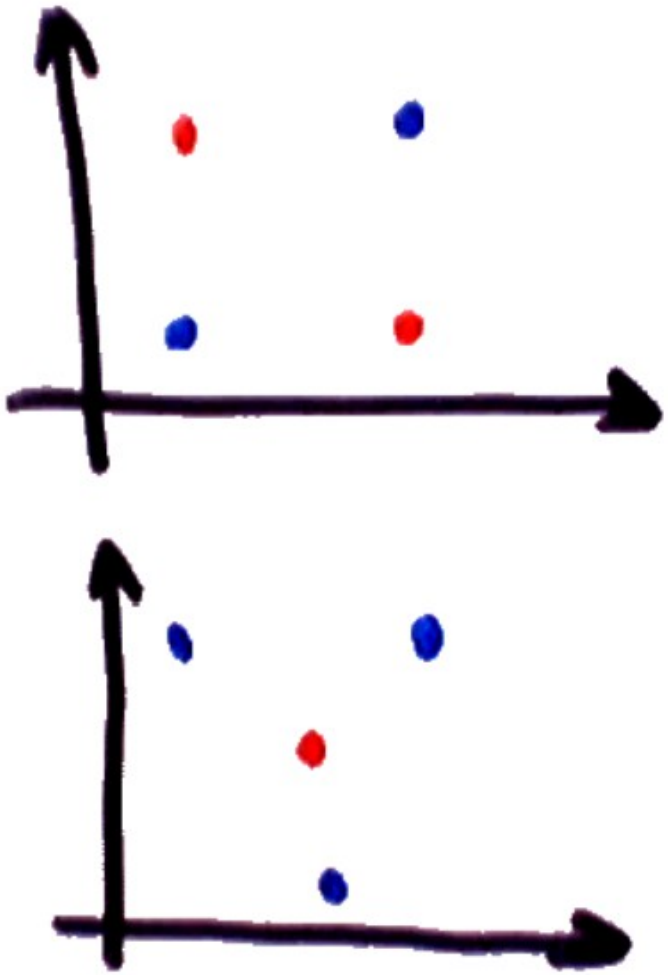
Number of Linearly Separable Dichotomies

The probability that P samples of dimension N are linearly separable goes to zero very quickly as P grows larger than N (Cover's theorem, 1966).



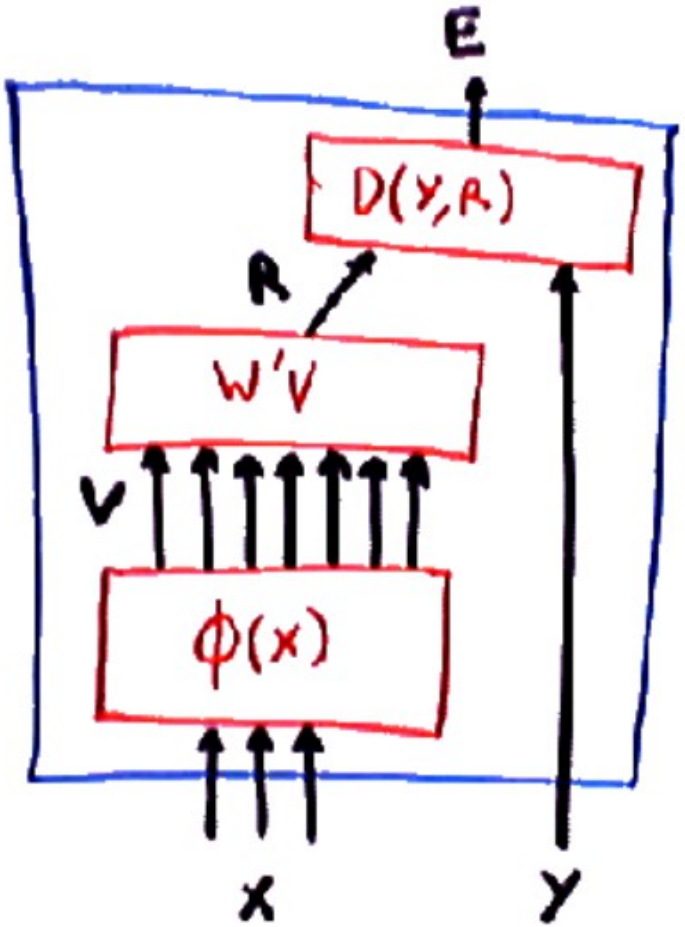
- Problem: there are 2^P possible dichotomies of P points.
- Only about N are linearly separable.
- If P is larger than N, the probability that a random dichotomy is linearly separable is very, very small.

Example of Non-Linearly Separable Dichotomies



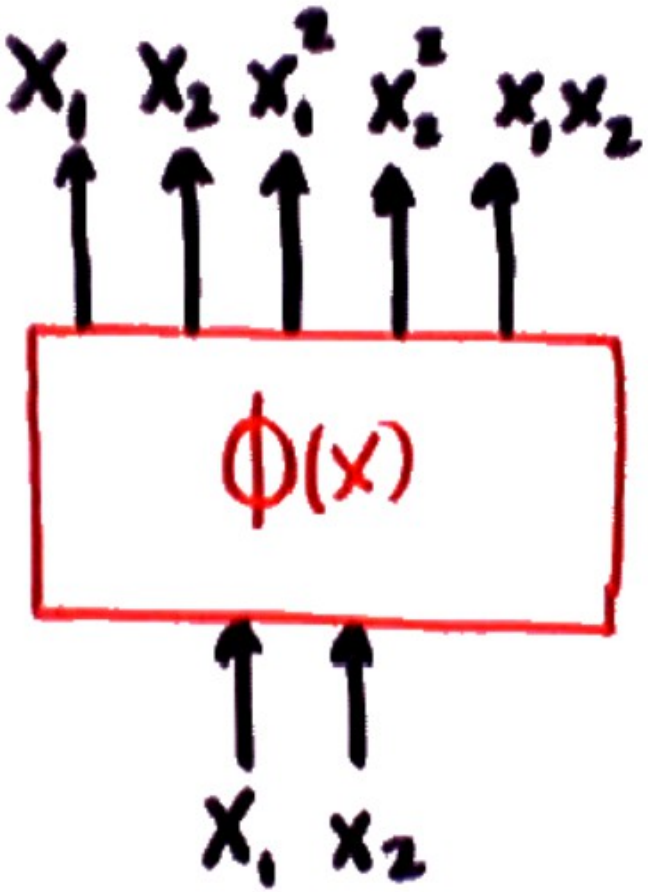
- Some seemingly simple dichotomies are not linearly separable
- Question: How do we make a given problem linearly separable?

Making N Larger: Preprocessing



- Answer 1: we make N larger by augmenting the input variables with new "features".
- we map/project X from its original N-dimensional space into a higher dimensional space where things are more likely to be linearly separable, using a vector function $\Phi(X)$.
- $E(Y, X, W) = D(Y, R)$
- $R = f(W'V)$
- $V = \Phi(X)$

Adding Cross-Product Terms

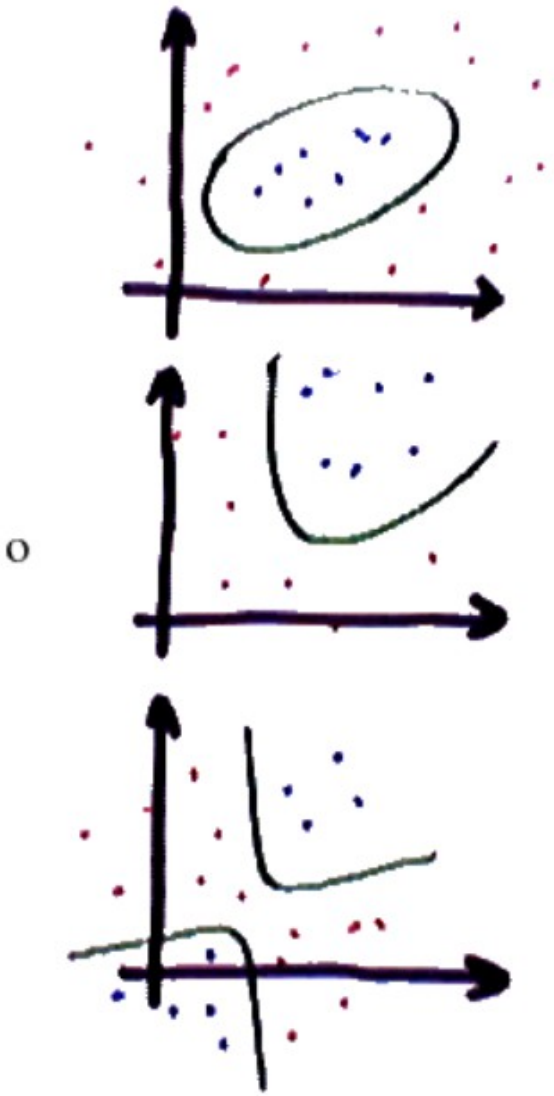


- Polynomial Expansion.
- If our original input variables are $(1, x_1, x_2)$, we construct a new *feature vector* with the following components:

$$\Phi(1, x_1, x_2) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

  i.e. we add all the cross-products of the original variables.
- we map/project X from its original N-dimensional space into a higher dimensional space with $N(N+1)/2$ dimensions.

Polynomial Mapping



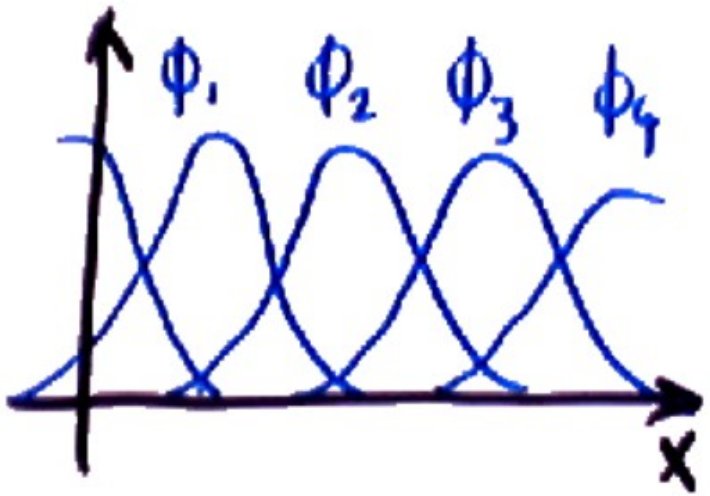
- Many new functions are now separable with the new architecture.
- With cross-product features, the family of class boundaries in the original space is the conic sections (ellipse, parabola, hyperbola).
- to each possible boundary in the original space corresponds a linear boundary in the transformed space.
- Because this is essentially a linear classifier with a preprocessing, we can use standard linear learning algorithms (perceptron, linear regression, logistic regression...).

Problems with Polynomial Mapping

- We can generalize this idea to higher degree polynomials, adding cross-product terms with 3, 4 or more variables.
- Unfortunately, the number of terms is the number of combinations d choose N, which grows like N^d, where d is the degree, and N the number of original variables.
- In particular, the number of free parameters that must be learned is also of order N^d.
- This is impractical for large N and for $d > 2$.
- Example: handwritten digit recognition (16x16 pixel images). Number of variables: 256. Degree 2: 32,896 variables. Degree 3: 2,796,160. Degre 4: 247,460,160.....

Next Idea: Tile the Space

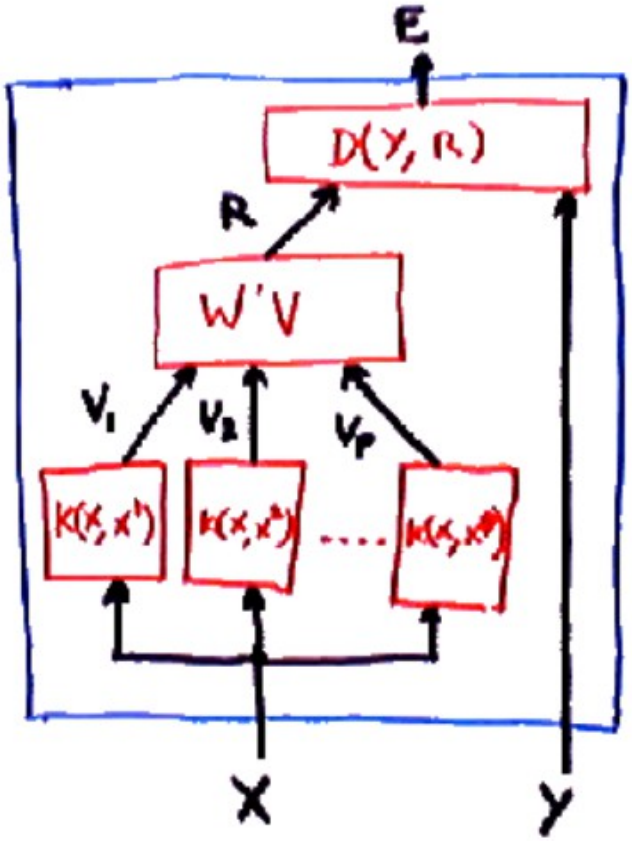
place a number of equally-spaced "bumps" that cover the entire input space.



- For classification, the bumps can be Gaussians
- For regression, the basis functions can be wavelets, sine/cosine, splines (pieces of polynomials)....
- problem: this does not work with more than a few dimensions.
- The number of bumps necessary to cover an N dimensional space grows exponentially with N.

Sample-Centered Basis Functions (Kernels)

Place the center of a basis function around each training sample. That way, we only spend resources on regions of the space where we actually have training samples.

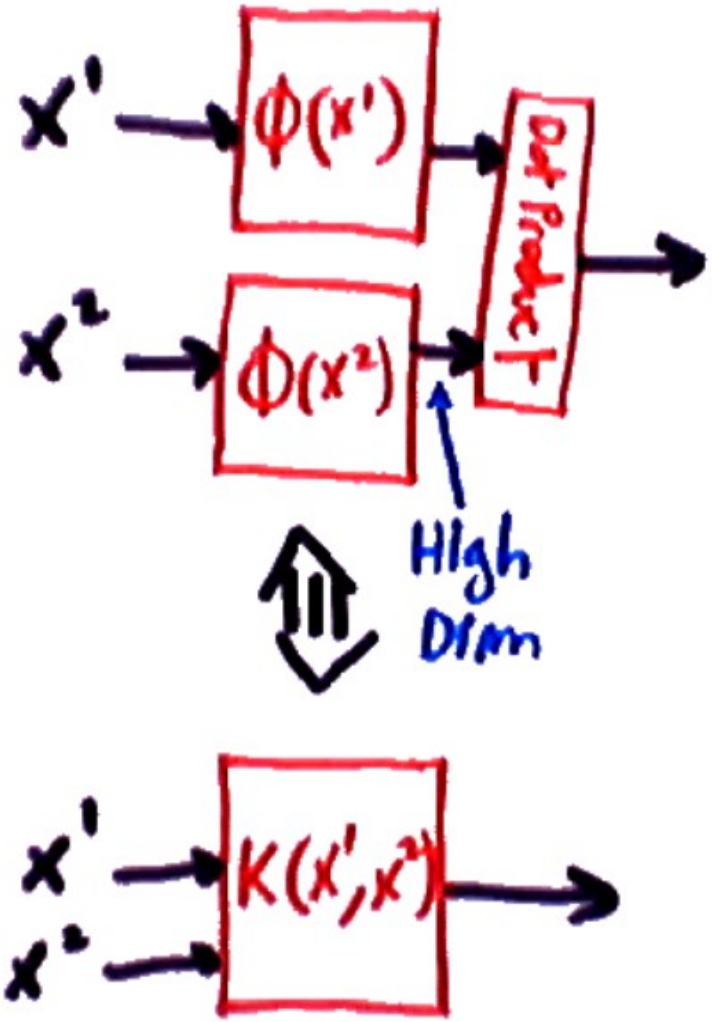


- Discriminant function:

$$f(X, W) = \sum_{k=1}^{k=P} W_k K(X, X^k)$$

- $K(X, X')$ often takes the form of a *radial basis function*: $K(X, X') = \exp(b||X - X'||^2)$ or a polynomial $K(X, X') = (X.X' + 1)^m$
- This is a very common architecture, which can be used with a number of energy functions.
- In particular, this is the architecture of the so-called Support Vector Machine (SVM), but the energy function of the SVM is a bit special. We will study it later in the course.

The Kernel Trick



- If the kernel function $K(X, X')$ verifies the *Mercer conditions*, then there exist a mapping Φ, such that $\Phi(X).\Phi(X') = K(X, X')$.
- The Mercer conditions are that K must be symmetric, and must be positive definite (i.e $K(X, X)$ must be positive for all X).
- In other words, if we want to map our X into a high-dimensional space (so as to make them linearly separable), and all we have to do in that space is compute dot products, we can take a shortcut and simply compute $K(X^1, X^2)$ without going through the high-dimensional space.
- This is called the "kernel trick". It is used in many so-called Kernel-based methods, including Support Vector Machines.

Examples of Kernels

- Quadratic kernel: $\Phi(X) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x2, x_1^2, x_2^2)$ then

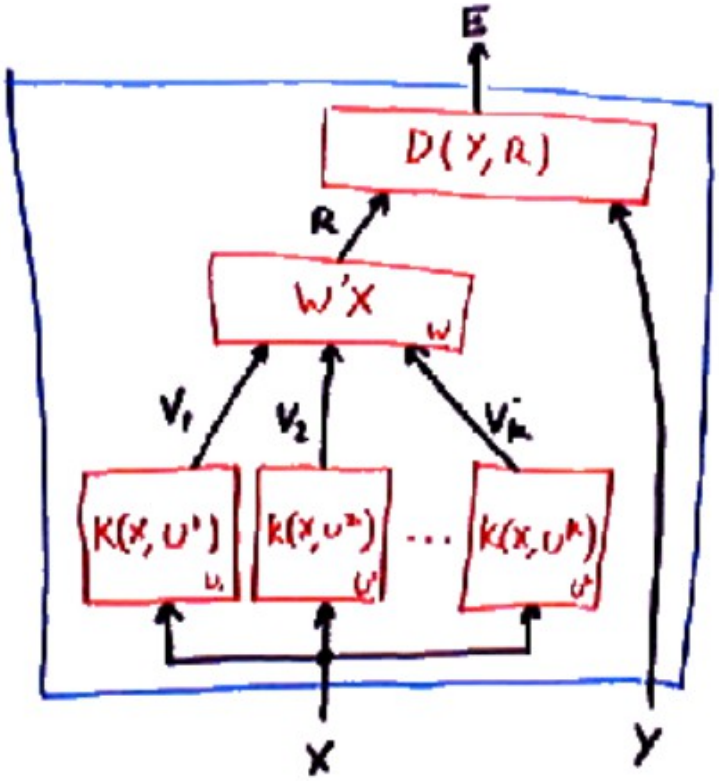
$$K(X, X') = \Phi(X).\Phi(X') = (X.X' + 1)^2$$

- Polynomial kernel: this generalizes to any degree d. The kernel that corresponds to $\Phi(X)$ bieng a polynomial of degree d is $K(X, X') = \Phi(X).\Phi(X') = (X.X' + 1)^d$.
- Gaussian Kernel:

$$K(X, X') = \exp(-b||X - X'||^2)$$

This kernel, sometimes called the Gaussian Radial Basis Function, is very commonly used.

Sparse Basis Functions

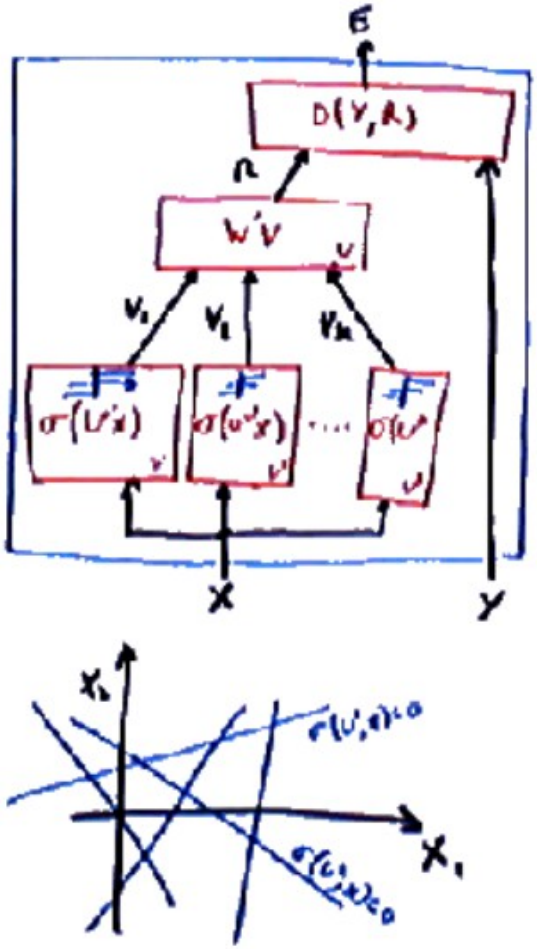


- Place the center of a basis function around areas containing training samples.
- Idea 1: use an unsupervised clustering algorithm (such as K-means or mixture of Gaussians) to place the centers of the basis functions in areas of high sample density.
- Idea 2: adjust the basis function centers through gradient descent in the loss function.

The discriminant function F is:

$$F(X, W, U^1, \ldots, U^K) = \sum_{k=1}^{k=K} W_k K(X, U^k)$$

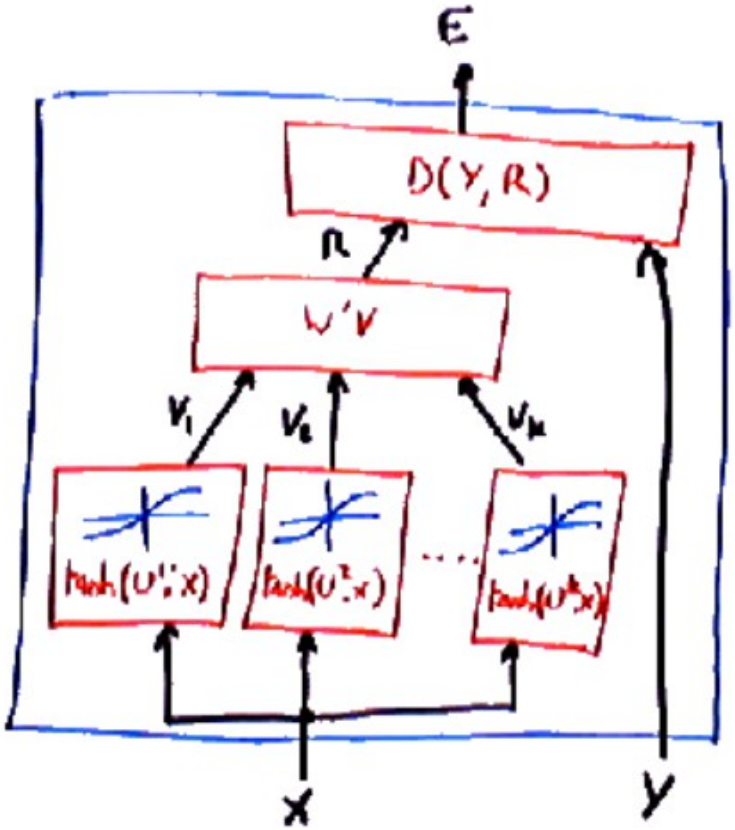
Other Idea: Random Directions



- Partition the space in lots of little domains by randomly placing lits of hyperplanes.
- Use many variables of the type $q(W^k X)$, where q is the threshold function (or some other squashing function) and W_k is a randomly picked vector.
- This is the original Perceptron.
- Without the non-linearity, the whole system would be linear (product of linear operations), and therefore would be no more powerful than a linear classifier.
- problem: a bit of a wishful thinking, but it works occasionally.

Neural Net with a Single Hidden Layer

A particularly interesting type of basis function is the sigmoid unit: $V_k = \tanh(U'^k X)$



- a network using these basis functions, whose output is $R = \sum_{k=1}^{k=K} W_k V_k$ is called a *single hidden-layer neural network*.
- Similarly to the RBF network, we can compute the gradient of the loss function with respect to the U^k:

$$\frac{\partial L(W)}{\partial U^j} = \frac{\partial L(W)}{\partial R} W_j \frac{\partial tanh(U_j' X)}{\partial U_j}$$

$$= \frac{\partial L(W)}{\partial R} W_j tanh'(U_j' X) X'$$

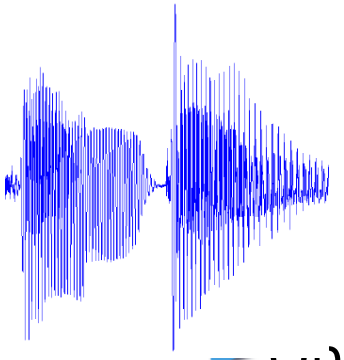
Any well-behaved function can be approximated as close as we wish by such networks (but K might be very large).

Architecture of "Mainstream"Pattern Recognition Systems

Y LeCun
MA Ranzato

Modern architecture for pattern recognition

- Speech recognition: early 90's – 2011



MFCC → Mix of Gaussians → Classifier

fixed — unsupervised — supervised

- Object Recognition: 2006 - 2012

SIFT / HoG → K-means / Sparse Coding → Pooling → Classifier

fixed — unsupervised — supervised

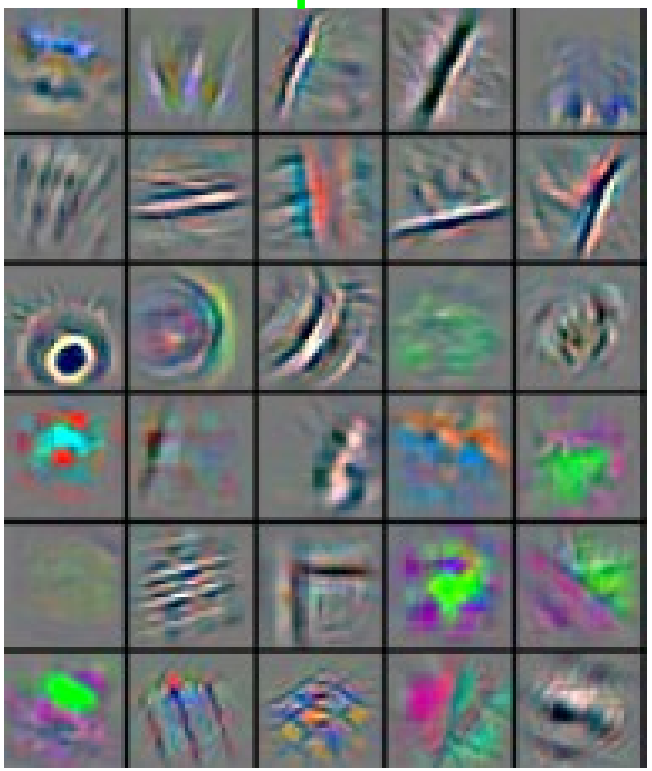
Low-level Features — Mid-level Features

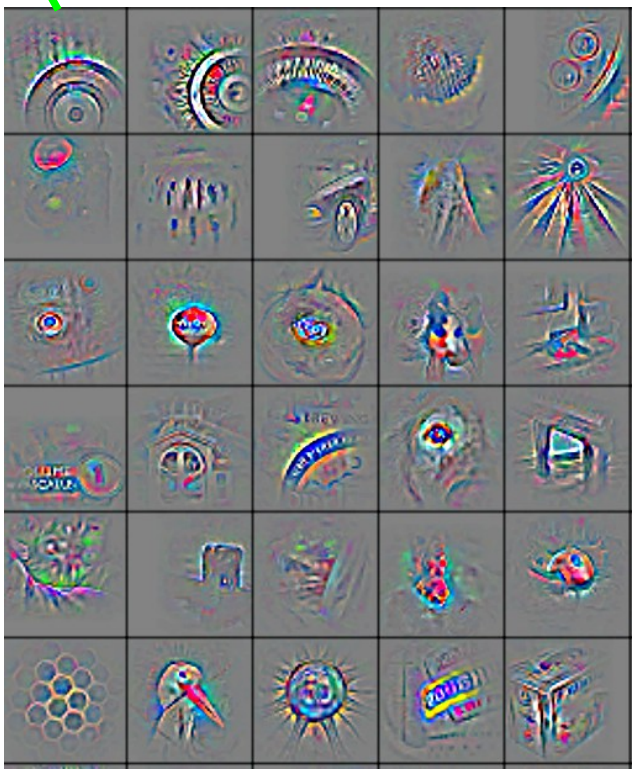
Deep Learning = Learning Hierarchical Representations

Y LeCun
MA Ranzato

- It's deep if it has more than one stage of non-linear feature transformation

Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier



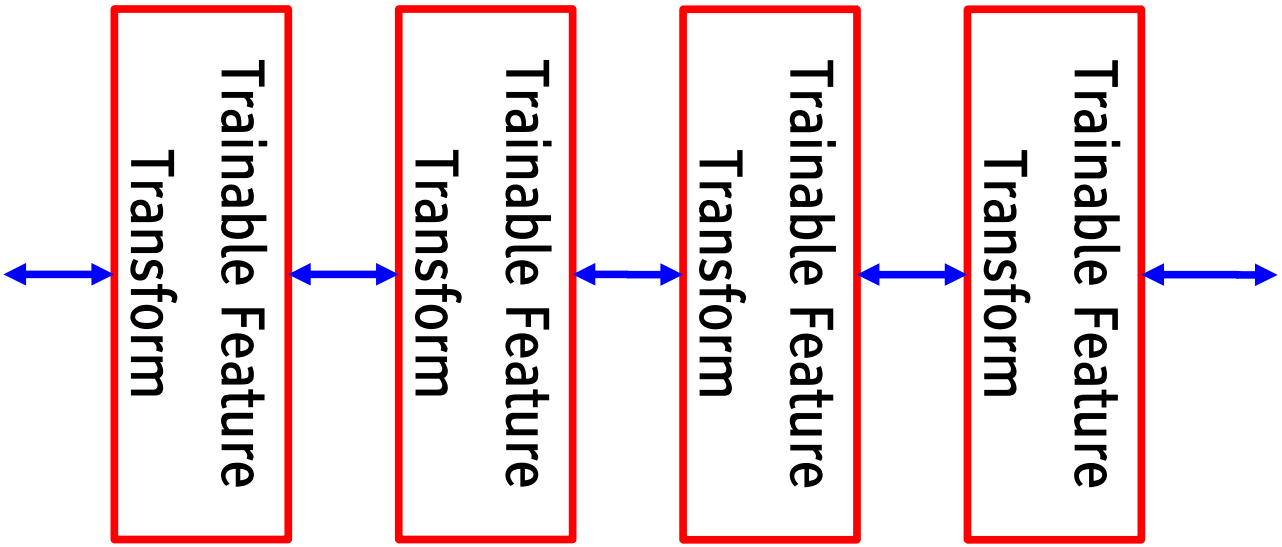


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Trainable Feature Hierarchy

Y LeCun
MA Ranzato

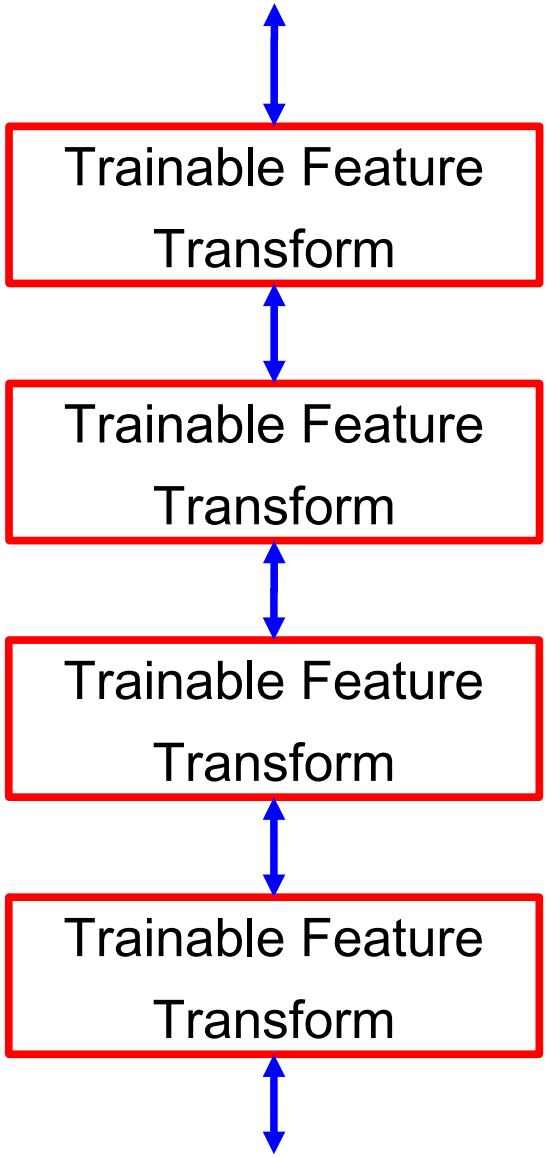
- **Hierarchy of representations with increasing level of abstraction**
- **Each stage is a kind of trainable feature transform**
- **Image recognition**
  - Pixel → edge → texton → motif → part → object
- **Text**
  - Character → word → word group → clause → sentence → story
- **Speech**
  - Sample → spectral band → sound → … → phone → phoneme → word



Learning Representations: a challenge for ML, CV, AI, Neuroscience, Cognitive Science...

Y LeCun
MA Ranzato

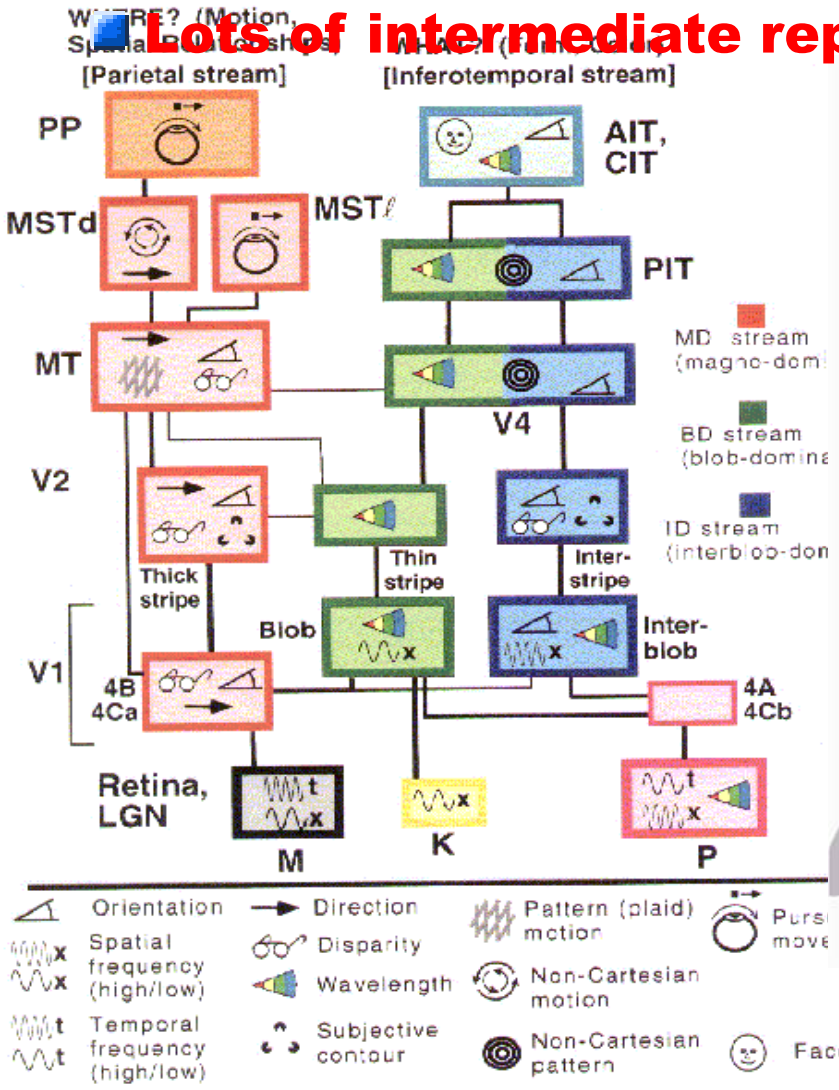
- **How do we learn representations of the perceptual world?**
  - How can a perceptual system build itself by looking at the world?
  - How much prior structure is necessary
- **ML/AI: how do we learn features or feature hierarchies?**
  - What is the fundamental principle? What is the learning algorithm? What is the architecture?
- **Neuroscience: how does the cortex learn perception?**
  - Does the cortex "run" a single, general learning algorithm? (or a small number of them)
- **CogSci: how does the mind learn abstract concepts on top of less abstract ones?**
- **Deep Learning addresses the problem of learning hierarchical representations with a single algorithm**



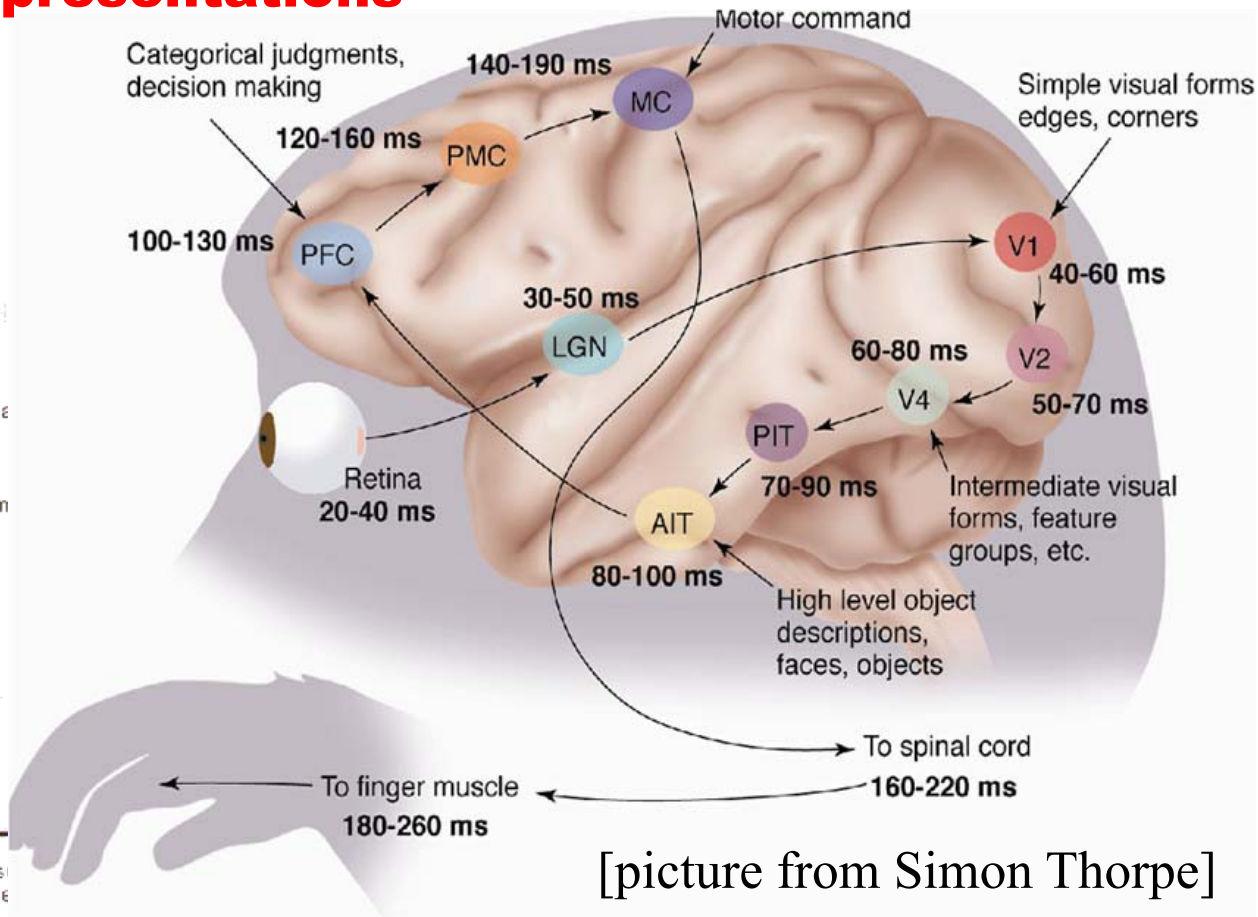
The Mammalian Visual Cortex is Hierarchical

Y LeCun
MA Ranzato

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations



[Gallant & Van Essen]



[picture from Simon Thorpe]

Let's be inspired by nature, but not too much

Y LeCun
MA Ranzato

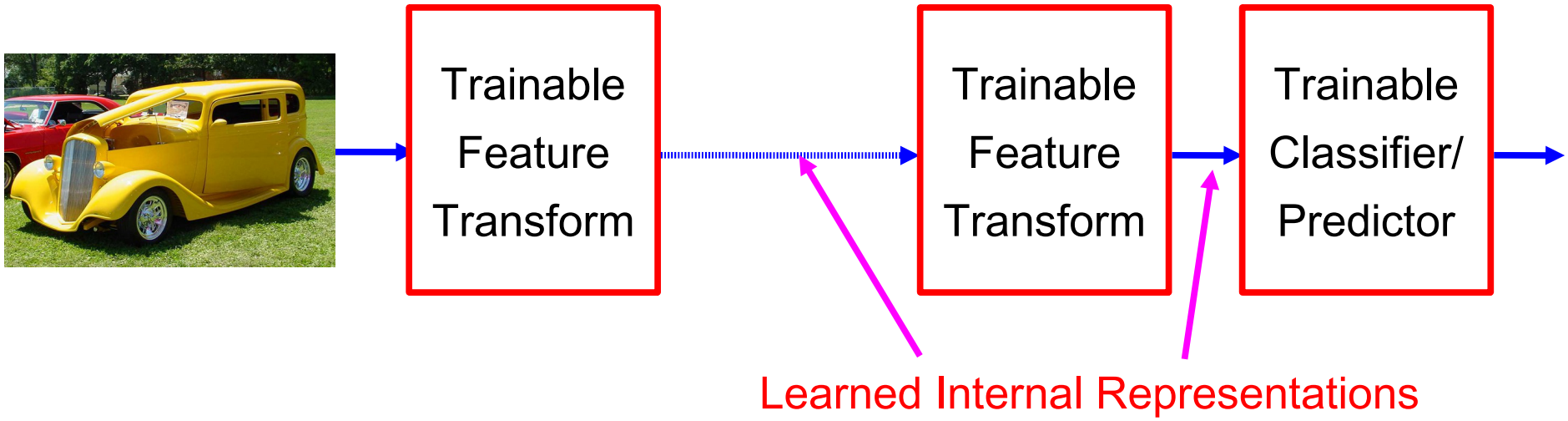
- **It's nice imitate Nature,**
- **But we also need to understand**
  - How do we know which details are important?
  - Which details are merely the result of evolution, and the constraints of biochemistry?
- **For airplanes, we developed aerodynamics and compressible fluid dynamics.**
  - We figured that feathers and wing flapping weren't crucial
- **QUESTION: What is the equivalent of aerodynamics for understanding intelligence?**

L'Avion III de Clément Ader, 1897
(Musée du CNAM, Paris)
His Eole took off from the ground in 1890, 13 years before the Wright Brothers, but you probably never heard of it.

Trainable Feature Hierarchies: End-to-end learning

Y LeCun
MA Ranzato

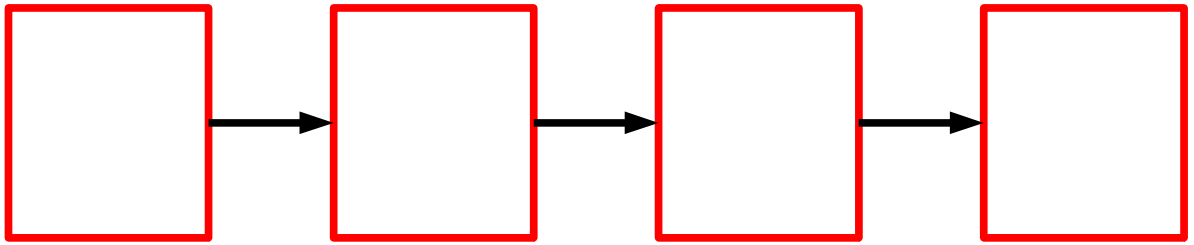
- **A hierarchy of trainable feature transforms**
  - Each module transforms its input representation into a higher-level one.
  - High-level features are more global and more invariant
  - Low-level features are shared among categories



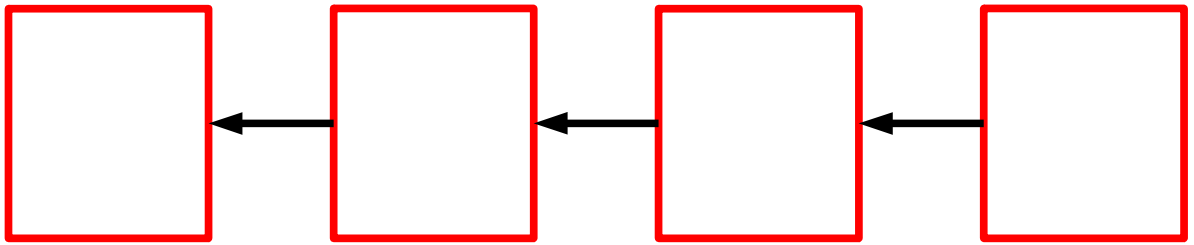
- How can we make all the modules trainable and get them to learn appropriate representations?

Three Types of Deep Architectures

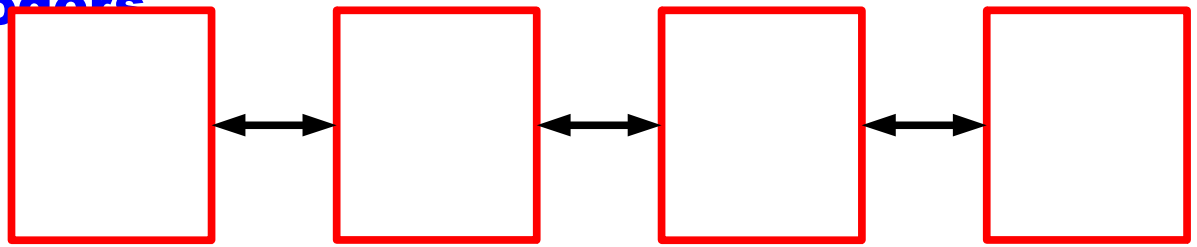
- **Feed-Forward: multilayer neural nets, convolutional nets**



- **Feed-Back: Stacked Sparse Coding, Deconvolutional Nets**



- **Bi-Drectional: Deep Boltzmann Machines, Stacked Auto-Encoders**



Three Types of Training Protocols

- **Purely Supervised**
  - Initialize parameters randomly
  - Train in supervised mode
    - typically with SGD, using backprop to compute gradients
  - Used in most practical systems for speech and image recognition
- **Unsupervised, layerwise + supervised classifier on top**
  - Train each layer unsupervised, one after the other
  - Train a supervised classifier on top, keeping the other layers fixed
  - Good when very few labeled samples are available
- **Unsupervised, layerwise + global supervised fine-tuning**
  - Train each layer unsupervised, one after the other
  - Add a classifier layer, and retrain the whole thing supervised
  - Good when label set is poor (e.g. pedestrian detection)
- **Unsupervised pre-training often uses regularized auto-encoders**

Do we really need deep architectures?

Y LeCun
MA Ranzato

- **Theoretician's dilemma: "We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?"**

$$y = \sum_{i=1}^{p} \alpha_i K(X, X^i) \qquad y = F(W^1.F(W^0.X))$$

  - kernel machines (and 2-layer neural nets) are "universal".

- **Deep learning machines**

$$y = F(W^K.F(W^{K-1}.F(....F(W^0.X)...)))$$

- **Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition**
  - they can represent more complex functions with less "hardware"
- **We need an efficient parameterization of the class of functions that are useful for "AI" tasks (vision, audition, NLP...)**

Why would deep architectures be more efficient?

[Bengio & LeCun 2007 "Scaling Learning Algorithms Towards AI"]

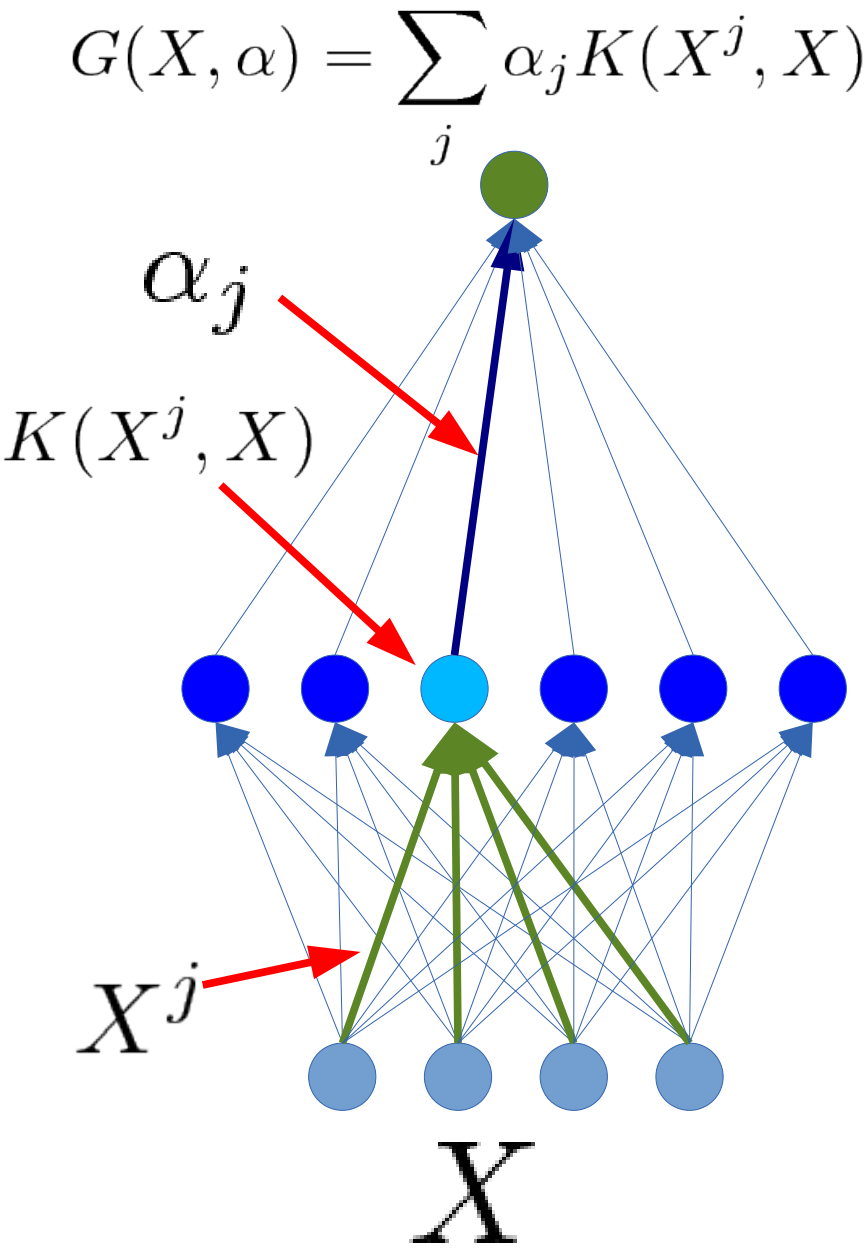
Y LeCun
MA Ranzato

- **A deep architecture trades space for time (or breadth for depth)**
  - more layers (more sequential computation),
  - but less hardware (less parallel computation).
- **Example1: N-bit parity**
  - requires N-1 XOR gates in a tree of depth log(N).
  - Even easier if we use threshold gates
  - requires an exponential number of gates of we restrict ourselves to 2 layers (DNF formula with exponential number of minterms).
- **Example2: circuit for addition of 2 N-bit binary numbers**
  - Requires O(N) gates, and O(N) layers using N one-bit adders with ripple carry propagation.
  - Requires lots of gates (some polynomial in N) if we restrict ourselves to two layers (e.g. Disjunctive Normal Form).
  - Bad news: almost all boolean functions have a DNF formula with an exponential number of minterms O(2^N).....

Which Models are Deep?

Y LeCun
MA Ranzato

- **2-layer models are not deep (even if you train the first layer)**
  - Because there is no feature hierarchy
- **Neural nets with 1 hidden layer are not deep**
- **SVMs and Kernel methods are not deep**
  - Layer1: kernels; layer2: linear
  - The first layer is "trained" in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
- **Classification trees are not deep**
  - No hierarchy of features. All decisions are made in the input space

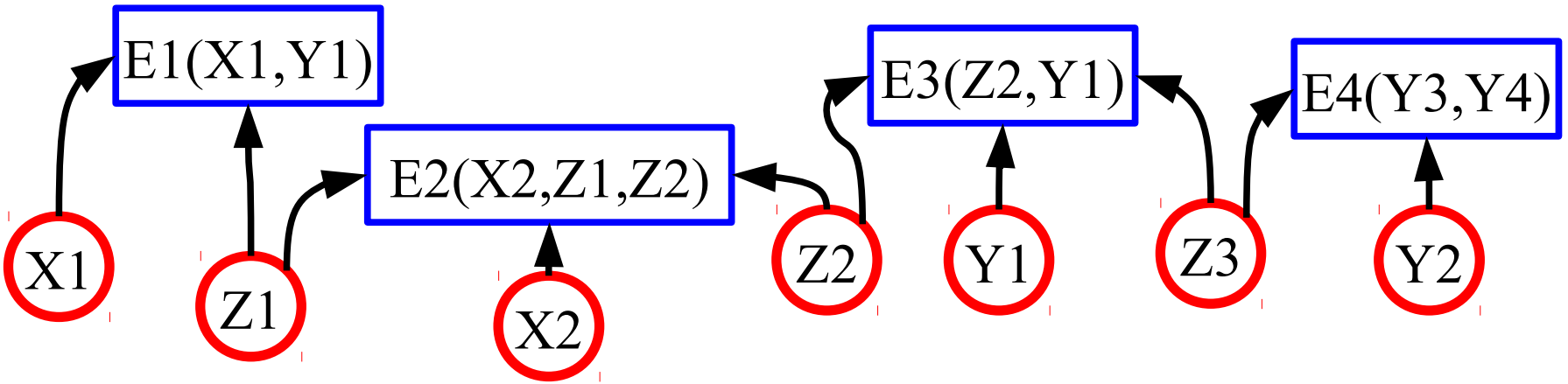


Are Graphical Models Deep?

Y LeCun
MA Ranzato

- **There is no opposition between graphical models and deep learning.**
  - Many deep learning models are formulated as factor graphs
  - Some graphical models use deep architectures inside their factors
- **Graphical models can be deep (but most are not).**
- **Factor Graph: sum of energy functions**
  - Over inputs X, outputs Y and latent variables Z. Trainable parameters: W

$$-\log P(X, Y, Z/W) \propto E(X, Y, Z, W) = \sum_i E_i(X, Y, Z, W_i)$$



- **Each energy function can contain a deep network**
- **The whole factor graph can be seen as a deep network**

Deep Learning: A Theoretician's Nightmare?

Y LeCun
MA Ranzato

- **Deep Learning involves non-convex loss functions**
  - With non-convex losses, all bets are off
  - Then again, every speech recognition system ever deployed has used non-convex optimization (GMMs are non convex).

- **But to some of us all "interesting" learning is non convex**
  - Convex learning is invariant to the order in which sample are presented (only depends on asymptotic sample frequencies).
  - Human learning isn't like that: we learn simple concepts before complex ones. The order in which we learn things matter.

Deep Learning: A Theoretician's Nightmare?

Y LeCun
MA Ranzato

- **No generalization bounds?**
  - Actually, the usual VC bounds apply: most deep learning systems have a finite VC dimension
  - We don't have tighter bounds than that.
  - But then again, how many bounds are tight enough to be useful for model selection?

- **It's hard to prove anything about deep learning systems**
  - Then again, if we only study models for which we can prove things, we wouldn't have speech, handwriting, and visual object recognition systems today.

Deep Learning: A Theoretician's Paradise?

Y LeCun
MA Ranzato

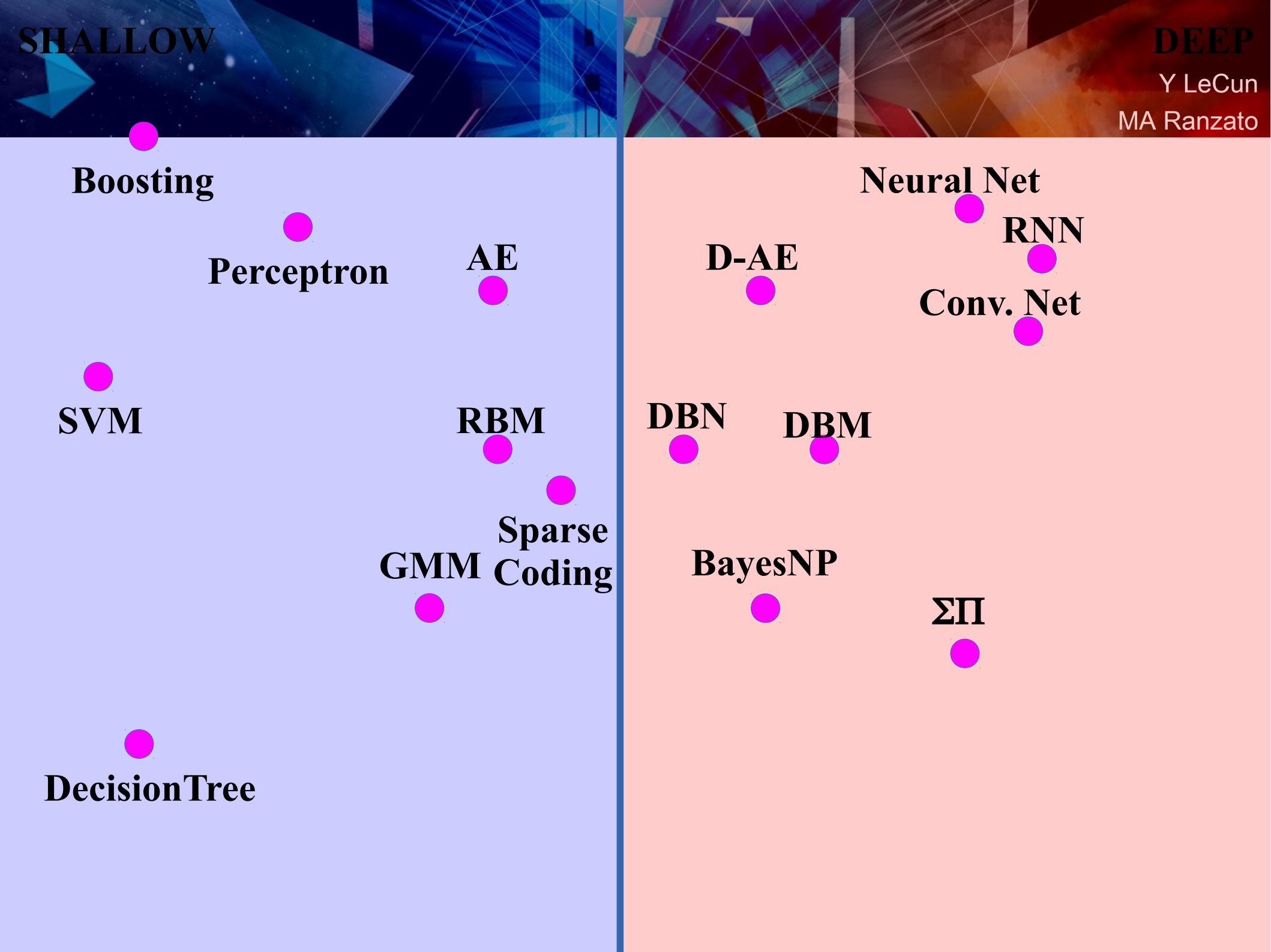
- **Deep Learning is about representing high-dimensional data**
  - There has to be interesting theoretical questions there
  - What is the geometry of natural signals?
  - Is there an equivalent of statistical learning theory for unsupervised learning?
  - What are good criteria on which to base unsupervised learning?
- **Deep Learning Systems are a form of latent variable factor graph**
  - Internal representations can be viewed as latent variables to be inferred, and deep belief networks are a particular type of latent variable models.
  - The most interesting deep belief nets have intractable loss functions: how do we get around that problem?

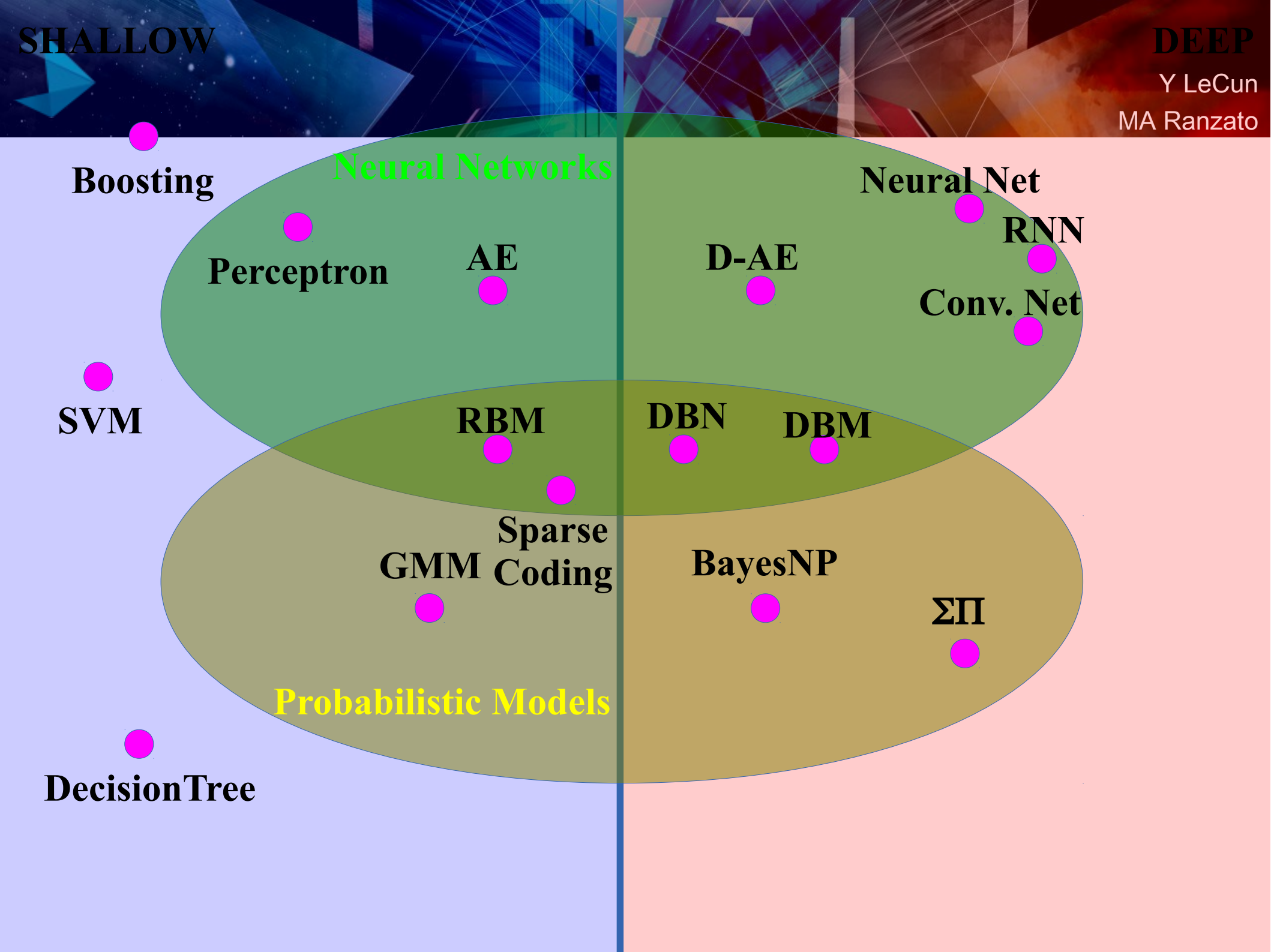
Deep Learning and Feature Learning Today

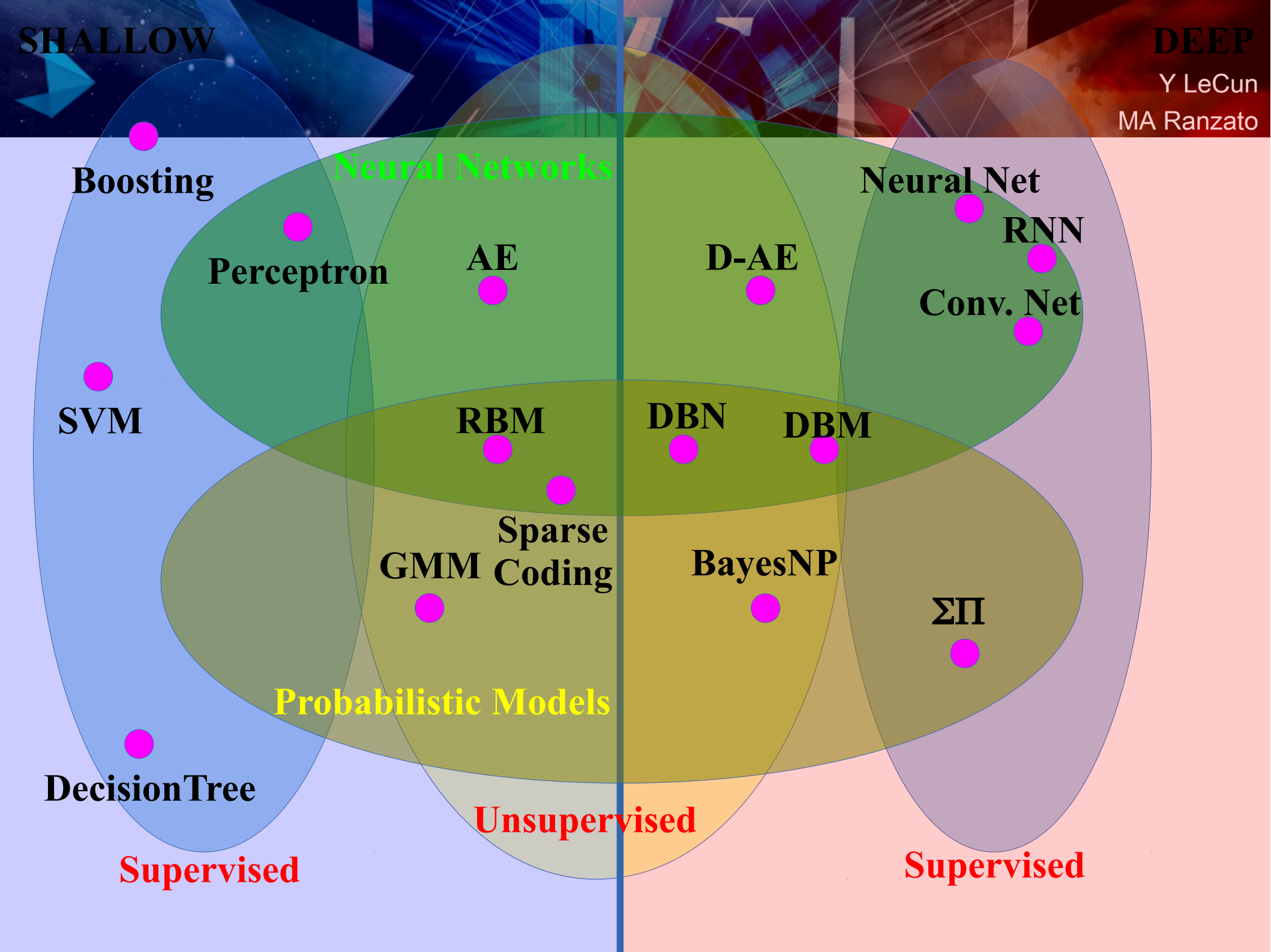
- **Deep Learning has been the hottest topic in speech recognition since 2010**
  - A few long-standing performance records were broken with deep learning methods
  - Microsoft and Google have both deployed DL-based speech recognition system in their products
  - Microsoft, Google, IBM, Nuance, Facebook, Baidu, and all the major academic and industrial players in speech recognition have projects on deep learning
- **Deep Learning is the hottest topic in Computer Vision**
  - Feature engineering is the bread-and-butter of a large portion of the CV community, which creates some resistance to feature learning
  - But the record holders on ImageNet and Semantic Segmentation are convolutional nets
- **Deep Learning is becoming hot in Natural Language Processing**
- **Deep Learning/Feature Learning in Applied Mathematics**
  - The connection with Applied Math is through sparse coding, non-convex optimization, stochastic gradient algorithms, etc...

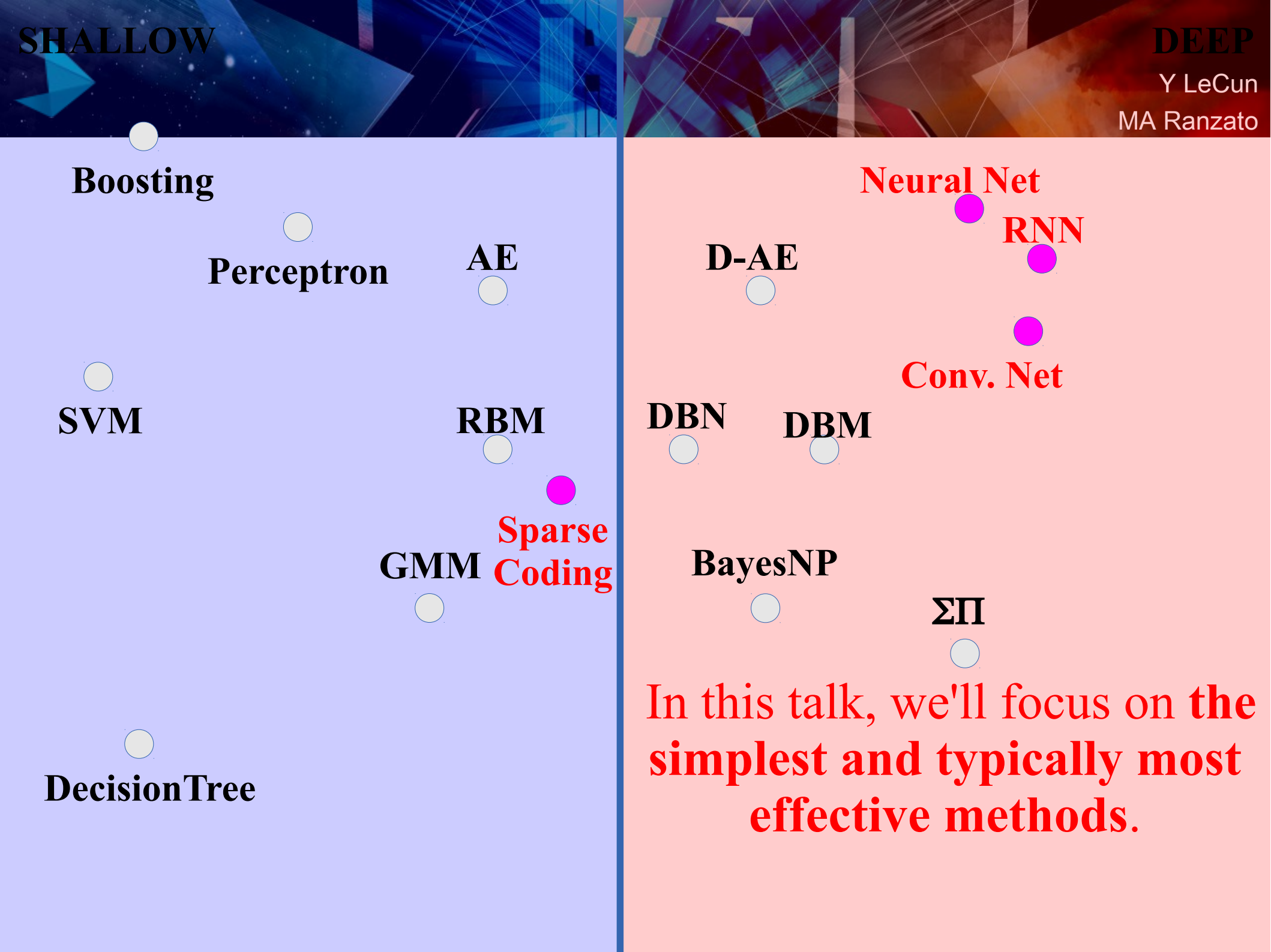
In Many Fields, Feature Learning Has Caused a Revolution (methods used in commercially deployed systems)

- **Speech Recognition I (late 1980s)**
  - Trained mid-level features with Gaussian mixtures (2-layer classifier)
- **Handwriting Recognition and OCR (late 1980s to mid 1990s)**
  - Supervised convolutional nets operating on pixels
- **Face & People Detection (early 1990s to mid 2000s)**
  - Supervised convolutional nets operating on pixels (YLC 1994, 2004, Garcia 2004)
  - Haar features generation/selection (Viola-Jones 2001)
- **Object Recognition I (mid-to-late 2000s: Ponce, Schmid, Yu, YLC....)**
  - Trainable mid-level features (K-means or sparse coding)
- **Low-Res Object Recognition: road signs, house numbers (early 2010's)**
  - Supervised convolutional net operating on pixels
- **Speech Recognition II (circa 2011)**
  - Deep neural nets for acoustic modeling
- **Object Recognition III, Semantic Labeling (2012, Hinton, YLC,...)**
  - Supervised convolutional nets operating on pixels


SHALLOW
DEEP
Y LeCun
MA Ranzato
Boosting
Perceptron
AE
SVM
RBM
Sparse Coding
GMM
DecisionTree
Neural Net
RNN
D-AE
Conv. Net
DBN
DBM
BayesNP
ΣΠ


SHALLOW
DEEP
Y LeCun
MA Ranzato
Boosting
Neural Networks
Neural Net
RNN
Perceptron
AE
D-AE
Conv. Net
SVM
RBM
DBN
DBM
Sparse
Coding
GMM
BayesNP
ΣΠ
Probabilistic Models
DecisionTree


SHALLOW
DEEP
Y LeCun
MA Ranzato
Boosting
Neural Networks
Neural Net
Perceptron
AE
D-AE
RNN
Conv. Net
SVM
RBM
DBN
DBM
Sparse Coding
GMM
BayesNP
ΣΠ
Probabilistic Models
DecisionTree
Unsupervised
Supervised
Supervised

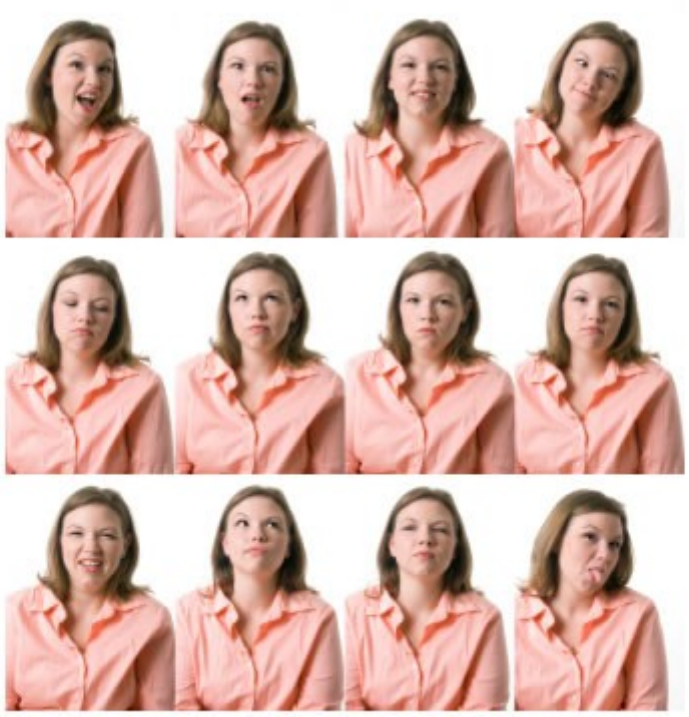

SHALLOW
DEEP
Y LeCun
MA Ranzato
Boosting
Perceptron
AE
SVM
RBM
Sparse Coding
GMM
DecisionTree
Neural Net
RNN
D-AE
Conv. Net
DBN
DBM
BayesNP
ΣΠ
In this talk, we'll focus on the simplest and typically most effective methods.

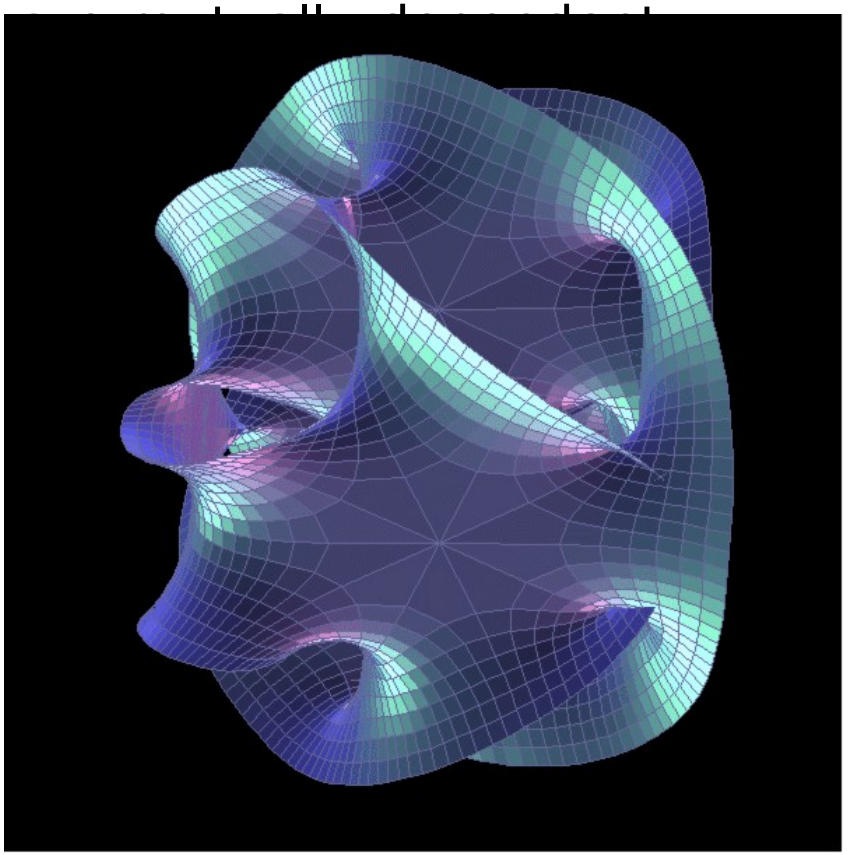

What Are
Good Feature?

Discovering the Hidden Structure in High-Dimensional Data
The manifold hypothesis

Y LeCun
MA Ranzato

- **Learning Representations of Data:**
  - **Discovering & disentangling the independent explanatory factors**
- **The Manifold Hypothesis:**
  - Natural data lives in a low-dimensional (non-linear) manifold
  - Because variables in natural data

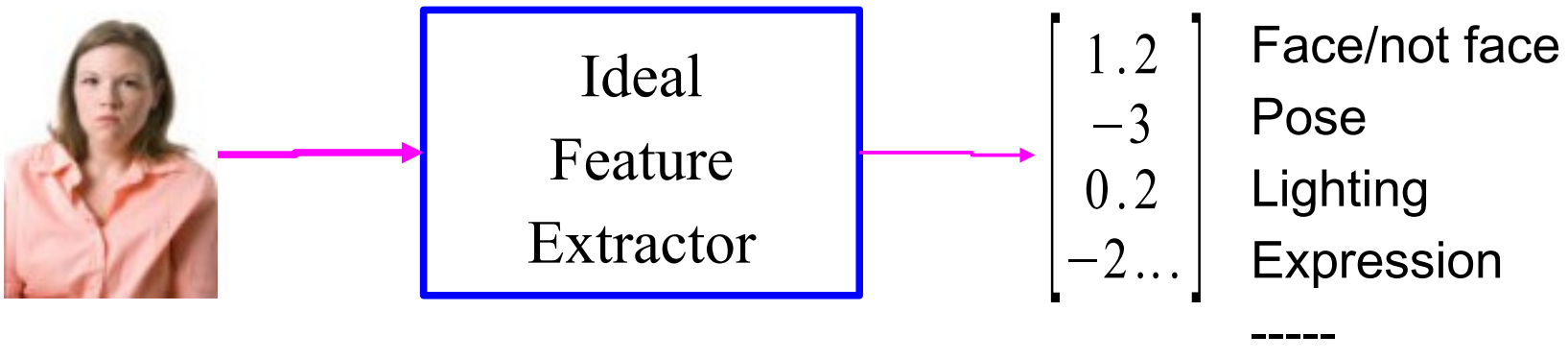




Discovering the Hidden Structure in High-Dimensional Data

Y LeCun
MA Ranzato

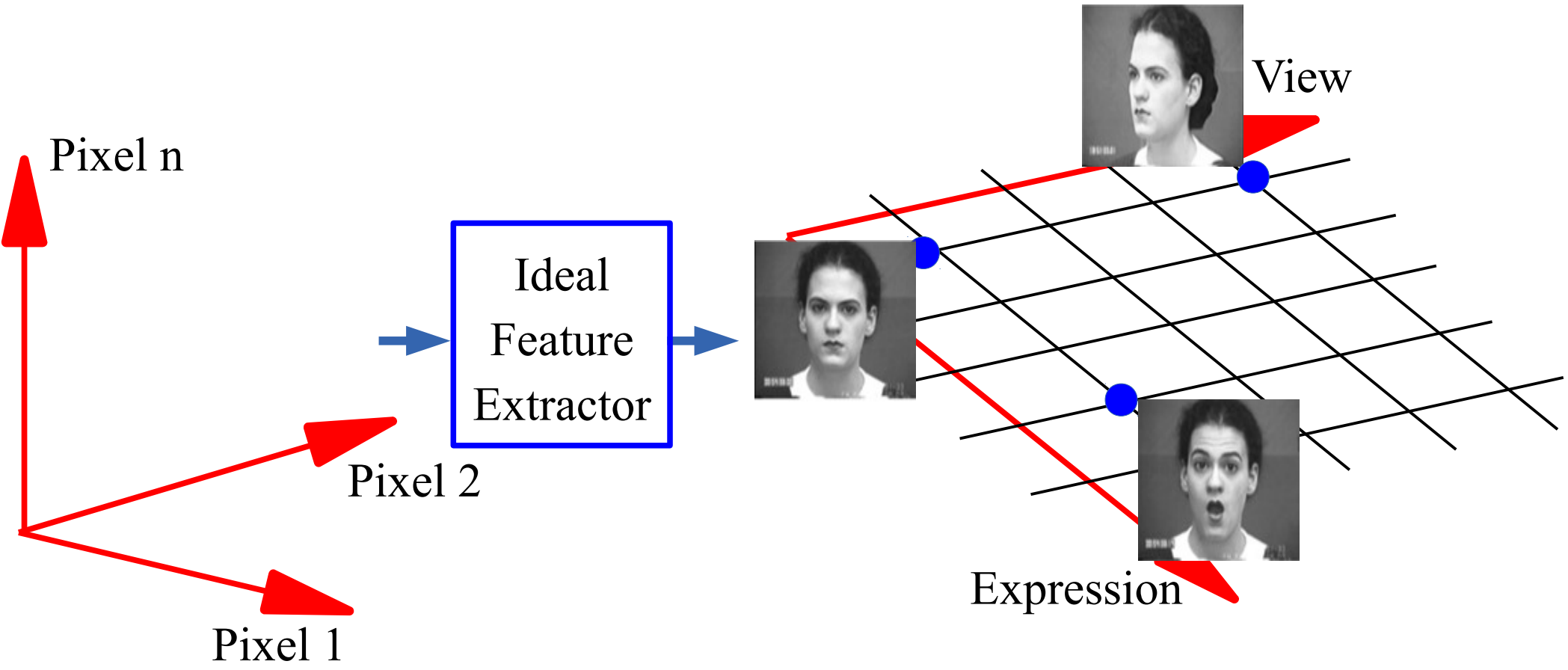
- **Example: all face images of a person**
  - 1000x1000 pixels = 1,000,000 dimensions
  - But the face has 3 cartesian coordinates and 3 Euler angles
  - And humans have less than about 50 muscles in the face
  - Hence the manifold of face images for a person has <56 dimensions
- **The perfect representations of a face image:**
  - Its coordinates on the face manifold
  - Its coordinates away from the manifold
- **We do not have good and general methods to learn functions that turns an image into this kind of representation**



Disentangling factors of variation

Y LeCun
MA Ranzato

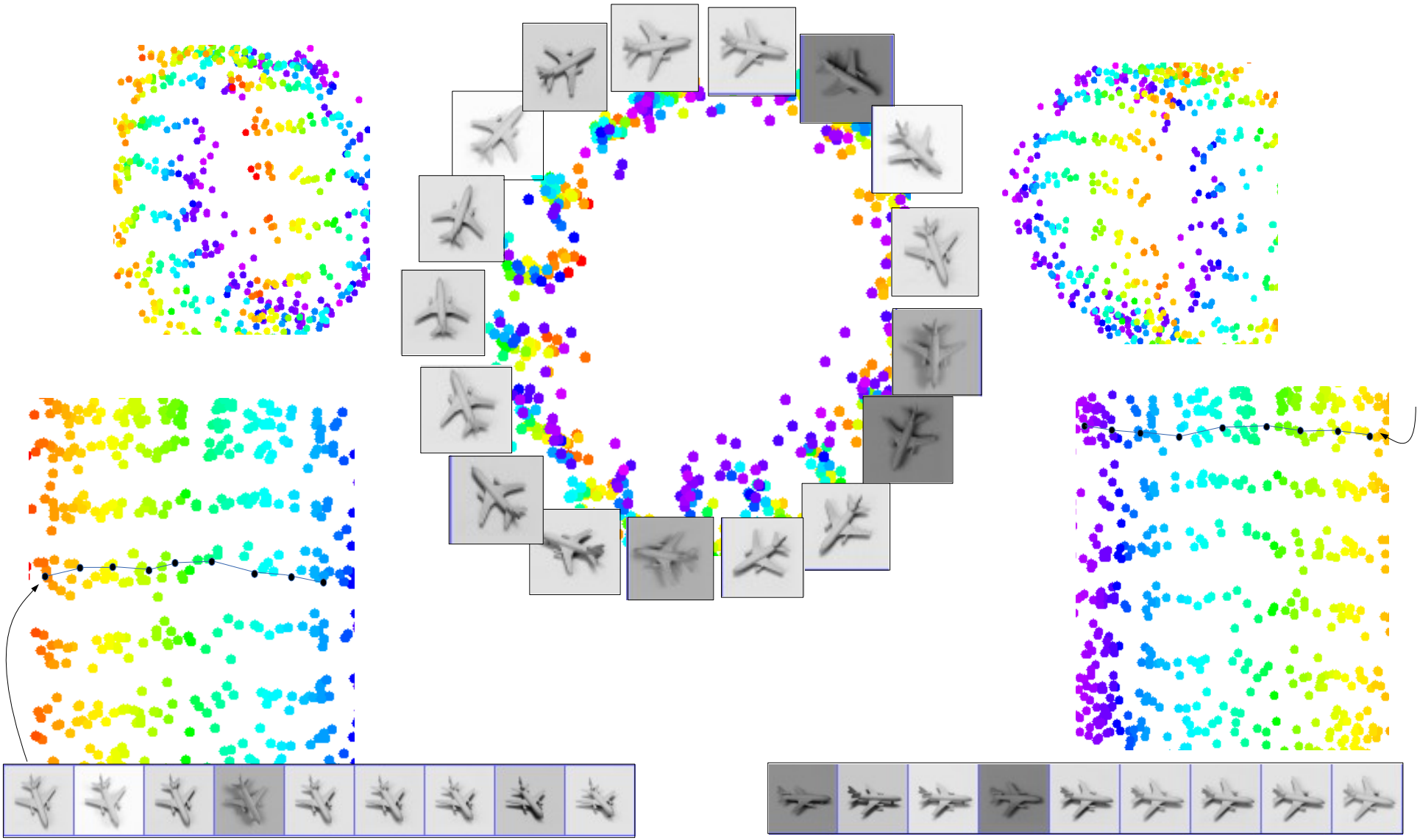
The Ideal Disentangling Feature Extractor



Data Manifold & Invariance: Some variations must be eliminated

Y LeCun
MA Ranzato

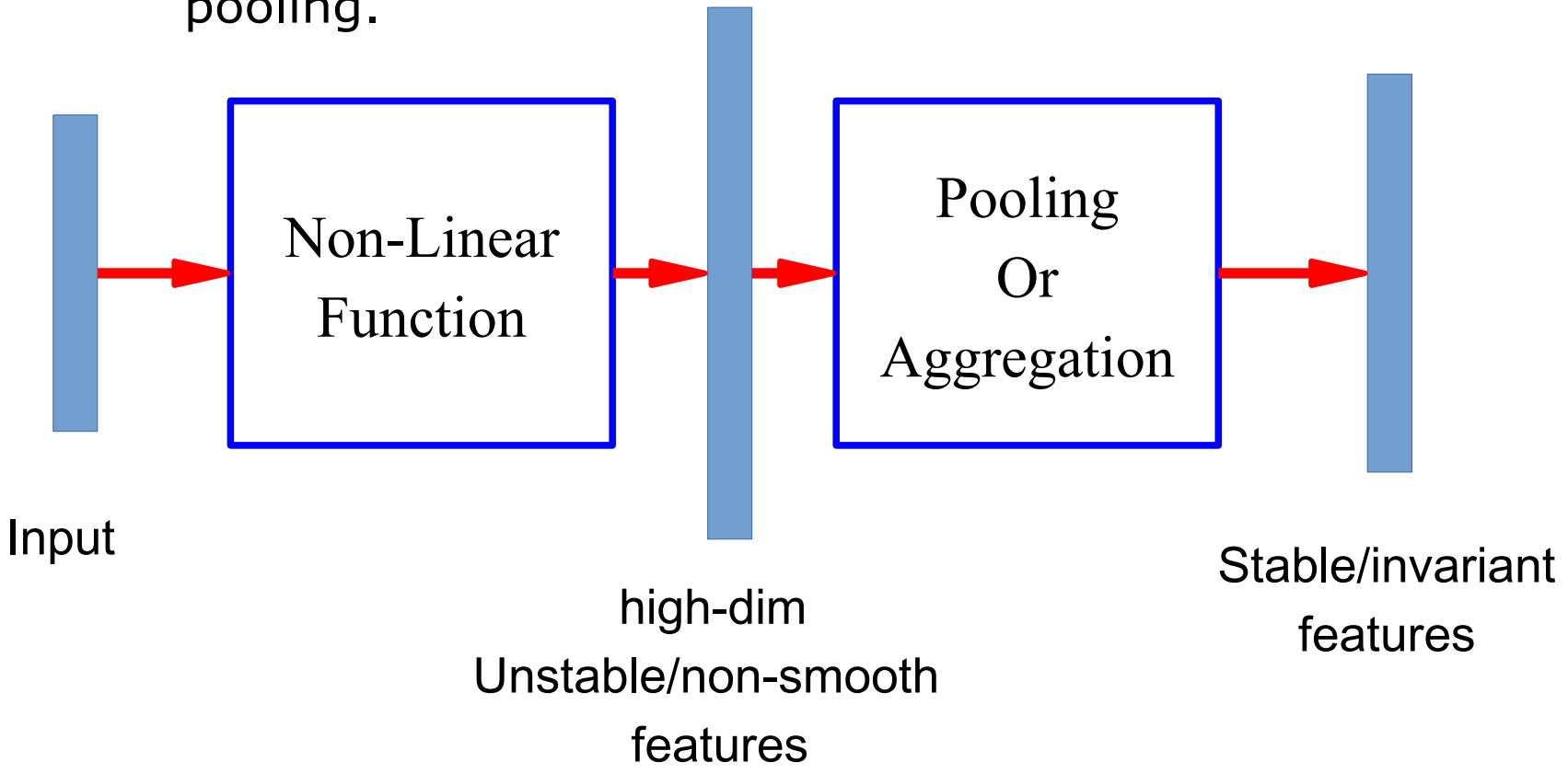
- **Azimuth-Elevation manifold. Ignores lighting.** [Hadsell et al. CVPR 2006]



Basic Idea for Invariant Feature Learning

Y LeCun
MA Ranzato

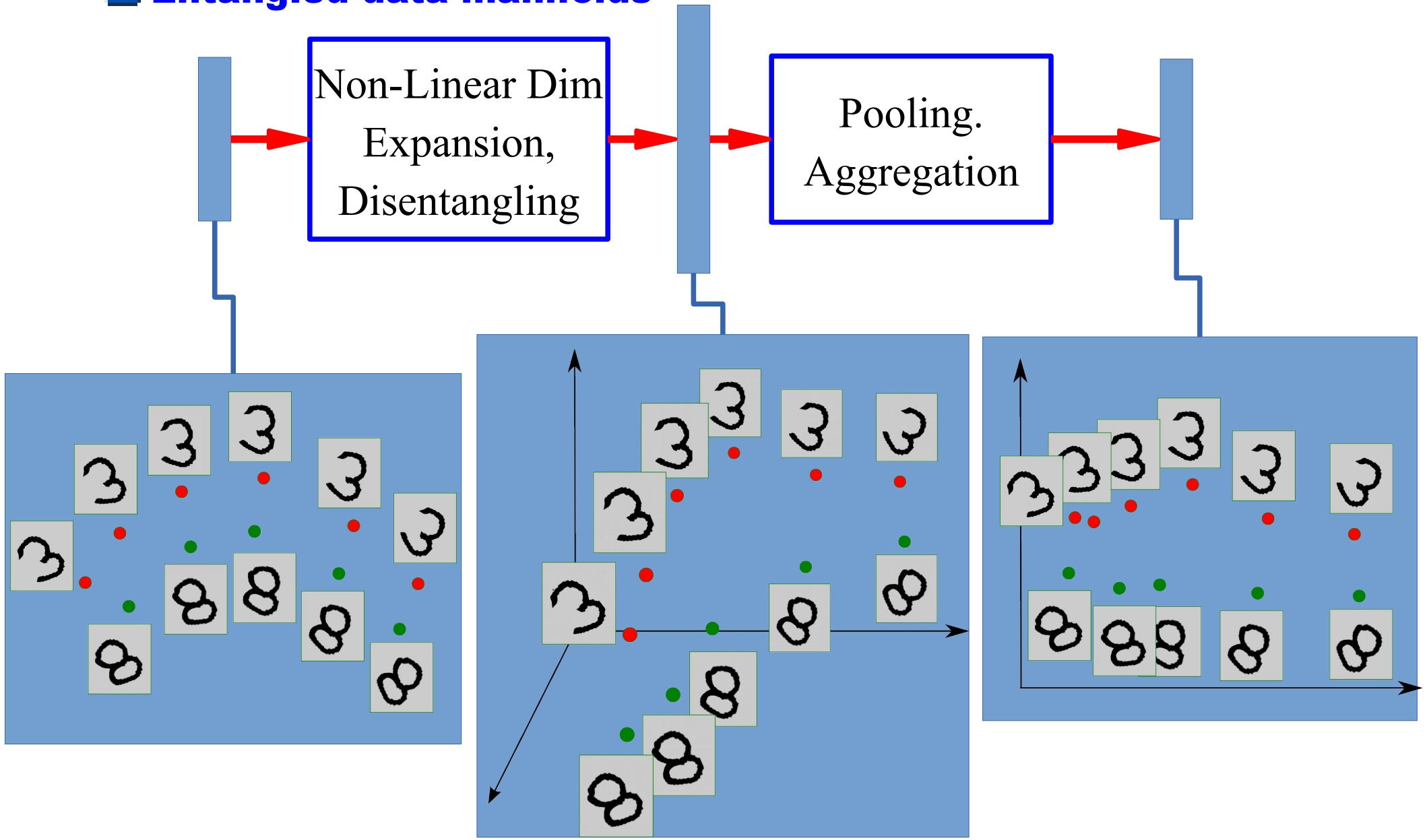
- **Embed the input non-linearly into a high(er) dimensional space**
  - In the new space, things that were non separable may become separable
- **Pool regions of the new space together**
  - Bringing together things that are semantically similar. Like pooling.



Non-Linear Expansion → Pooling

Y LeCun
MA Ranzato

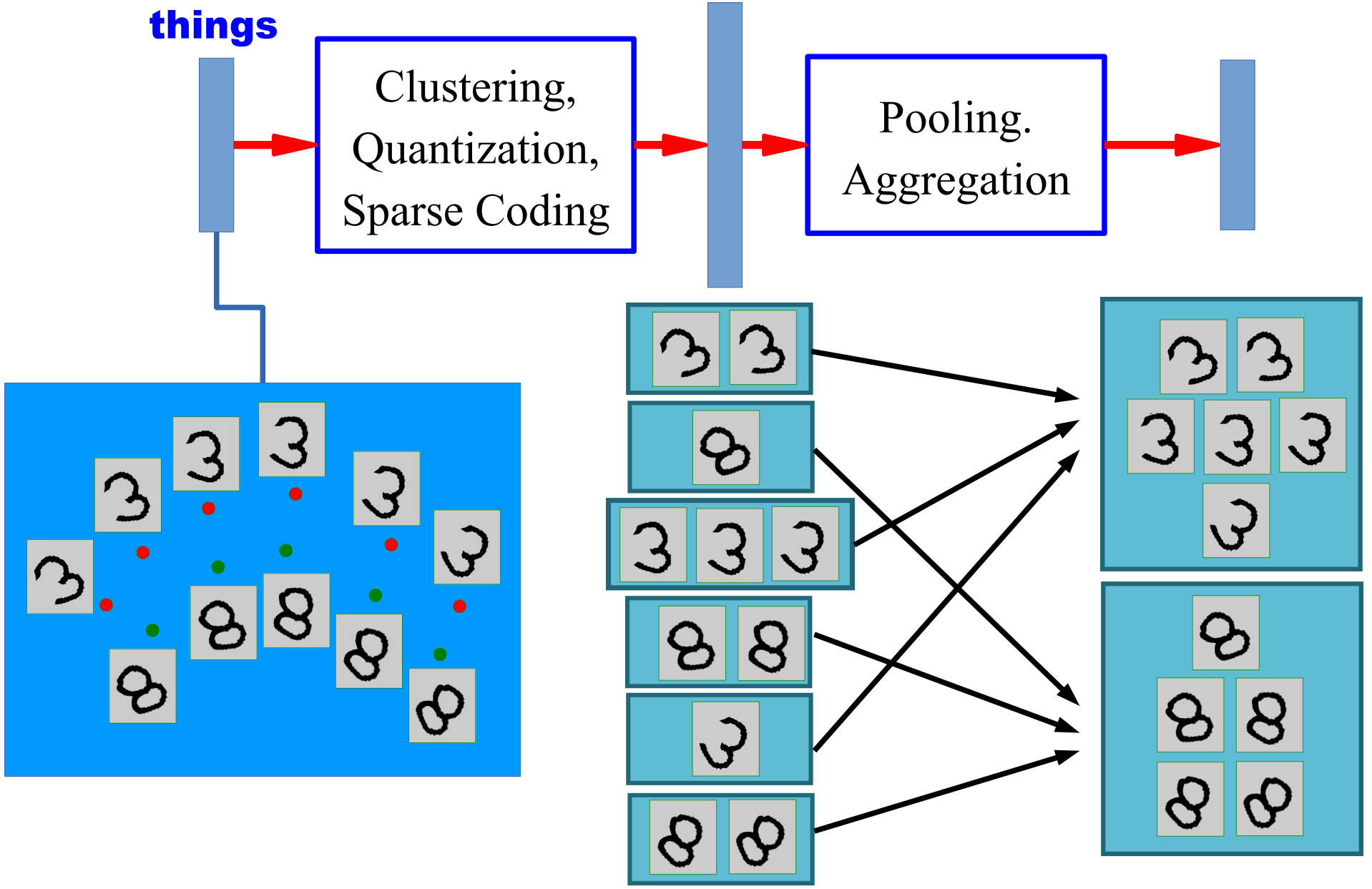
Entangled data manifolds



Sparse Non-Linear Expansion → Pooling

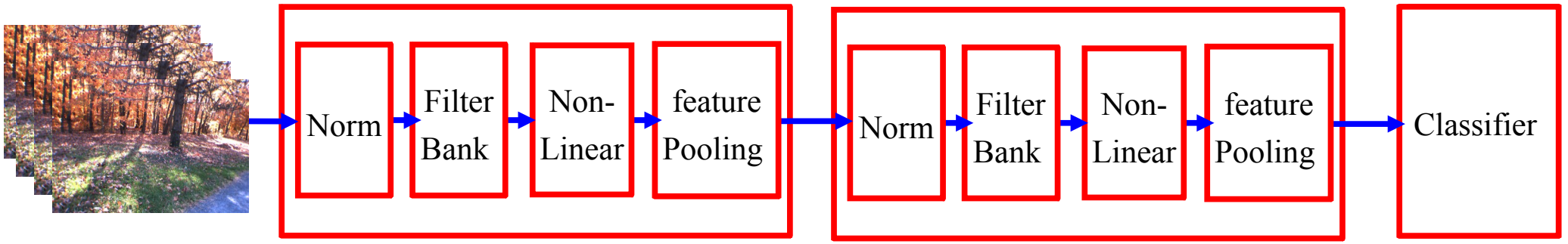
Y LeCun
MA Ranzato

- **Use clustering to break things apart, pool together similar things**



Overall Architecture: Normalization → Filter Bank → Non-Linearity → Pooling

Y LeCun
MA Ranzato

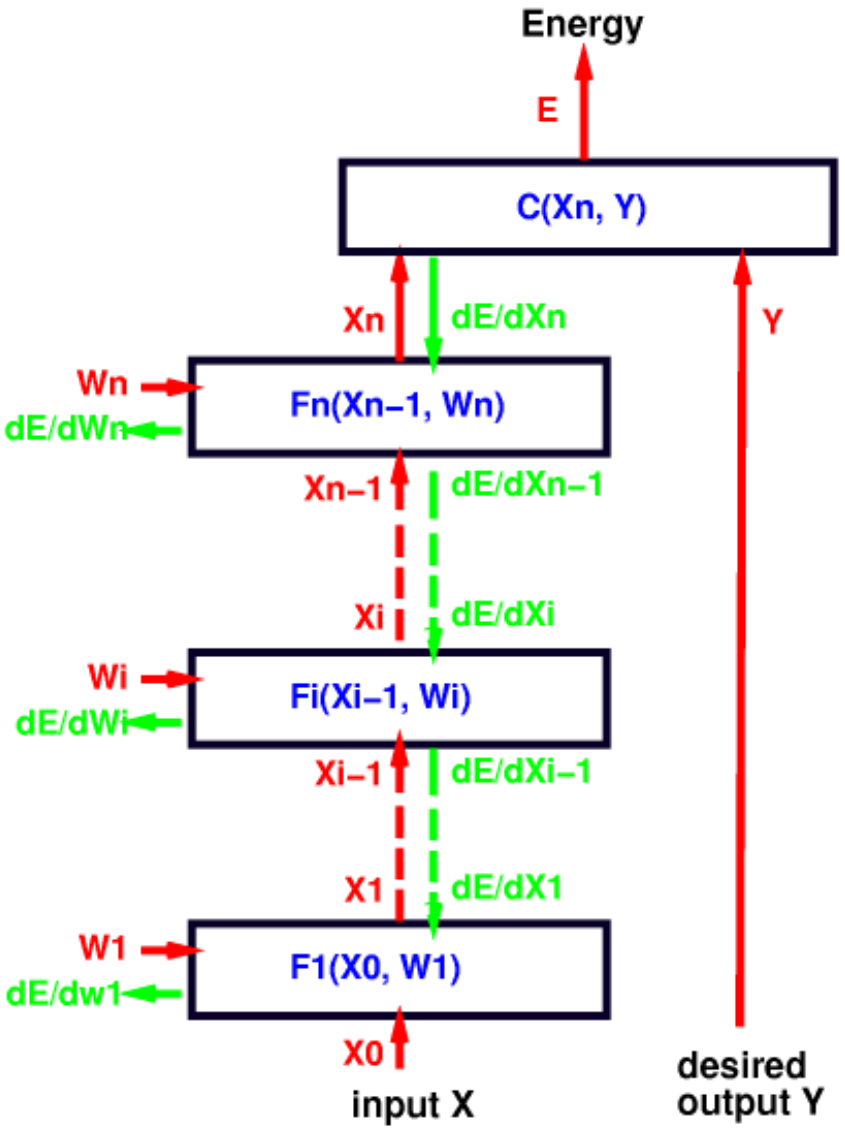


- **Stacking multiple stages of**
  - [Normalization → Filter Bank → Non-Linearity → Pooling].
- **Normalization: variations on whitening**
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank: dimension expansion, projection on overcomplete basis**
- **Non-Linearity: sparsification, saturation, lateral inhibition....**
  - Rectification (ReLU), Component-wise shrinkage, tanh, winner-takes-all
- **Pooling: aggregation over space or feature type**
  - $X_i; \quad L_p: \sqrt[p]{X_i^p}; \quad PROB: \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$


Deep Supervised Learning
(modular approach)

Multimodule Systems: Cascade

Y LeCun
MA Ranzato



- **Complex learning machines can be built by assembling modules into networks**
- **Simple example: sequential/layered feed-forward architecture (cascade)**
- **Forward Propagation:**

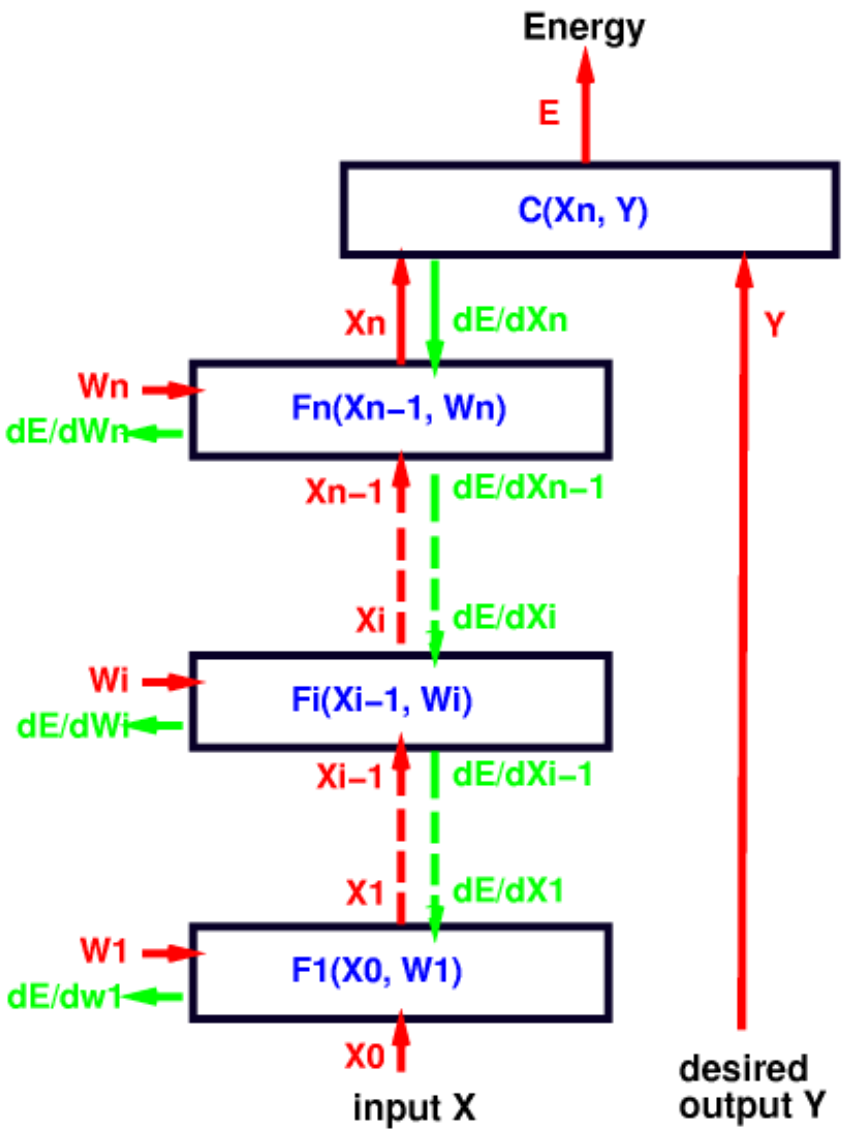
let $X = X_0$,

$$X_i = F_i(X_{i-1}, W_i) \quad \forall i \in [1, n]$$

$$E(Y, X, W) = C(X_n, Y)$$

Multimodule Systems: Implementation

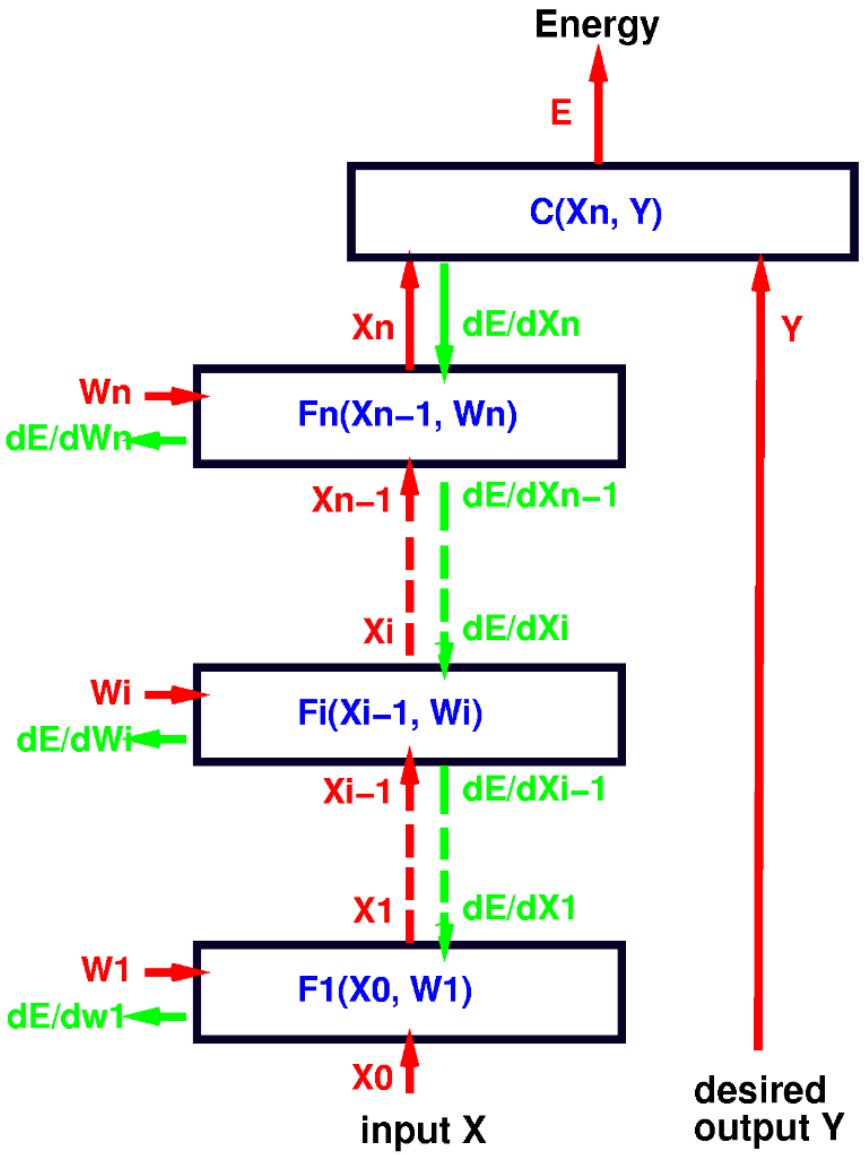
Y LeCun
MA Ranzato



- **Each module is an object**
  - Contains trainable parameters
  - Inputs are arguments
  - Output is returned, but also stored internally
  - Example: 2 modules m1, m2
- **Torch7 (by hand)**
  - hid = m1:forward(in)
  - out = m2:forward(hid)
- **Torch7 (using the nn.Sequential class)**
  - model = nn.Sequential()
  - model:add(m1)
  - model:add(m2)
  - out = model:forward(in)

Computing the Gradient in Multi-Layer Systems

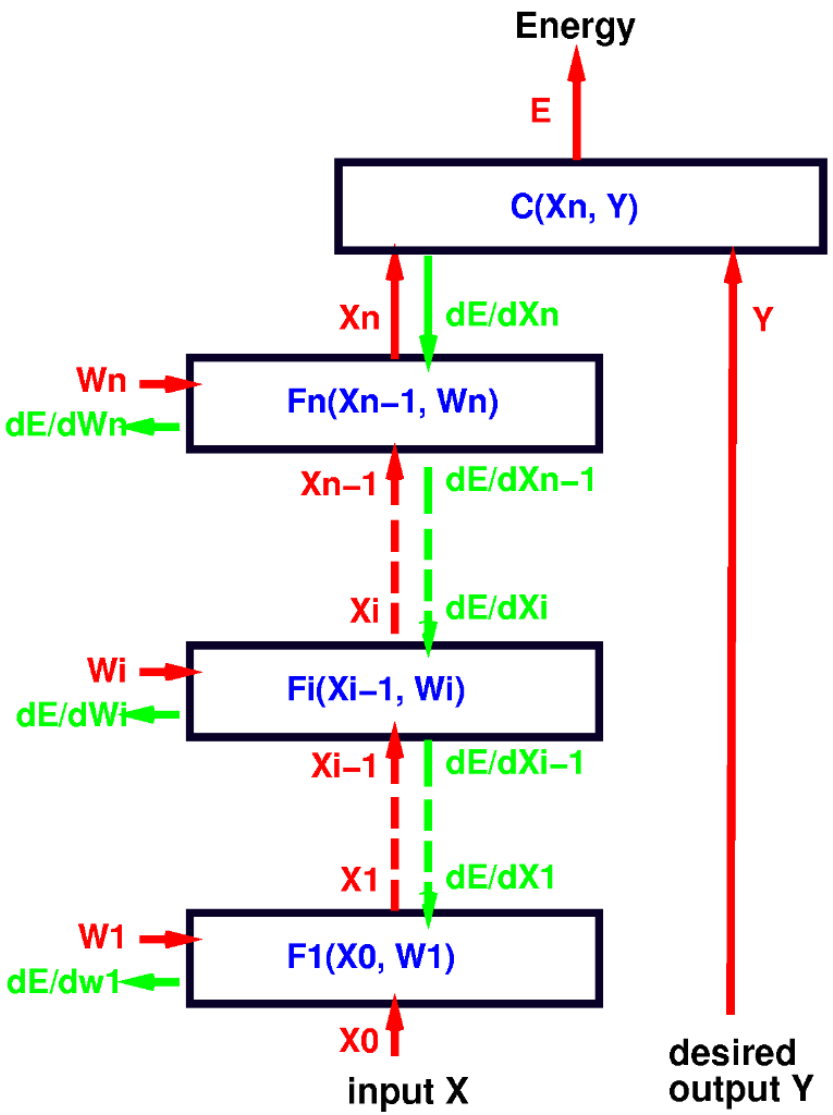
Y LeCun
MA Ranzato



- To train a multi-module system, we must compute the gradient of $E(W, Y, X)$ with respect to all the parameters in the system (all the W_k).
- Let's consider module i whose fprop method computes $X_k = F_k(X_{k-1}, W_k)$.
- Let's assume that we already know $\frac{\partial E}{\partial X_k}$, in other words, for each component of vector X_k we know how much E would wiggle if we wiggled that component of X_k.

Computing the Gradient in Multi-Layer Systems

Y LeCun
MA Ranzato



- We can apply chain rule to compute $\frac{\partial E}{\partial W_k}$ (how much E would wiggle if we wiggled each component of W_k):

$$\frac{\partial E}{\partial W_k} = \frac{\partial E}{\partial X_k} \frac{\partial F_k(X_{k-1}, W_k)}{\partial W_k}$$

$$[1 \times N_w] = [1 \times N_x].[N_x \times N_w]$$

- $\frac{\partial F_k(X_{k-1}, W_k)}{\partial W_k}$ is the *Jacobian matrix* of F_k with respect to W_k.

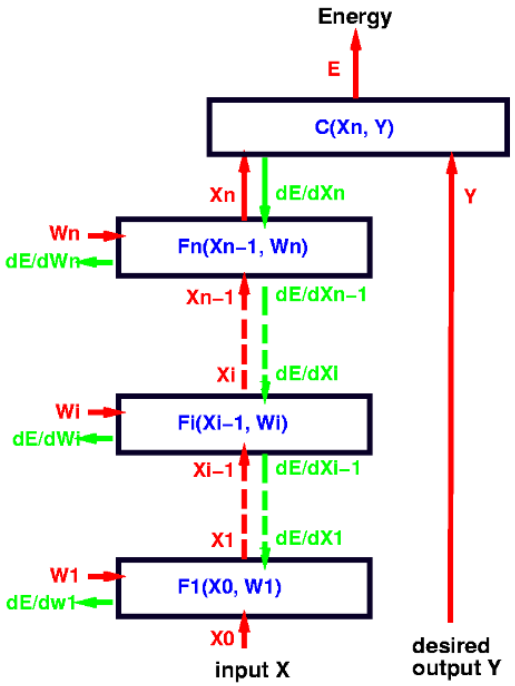
$$\left[\frac{\partial F_k(X_{k-1}, W_k)}{\partial W_k}\right]_{pq} = \frac{\partial \left[F_k(X_{k-1}, W_k)\right]_p}{\partial [W_k]_q}$$

- Element (p, q) of the Jacobian indicates how much the p-th output wiggles when we wiggle the q-th weight.

Computing the Gradient in Multi-Layer Systems

Y LeCun
MA Ranzato

Using the same trick, we can compute $\frac{\partial E}{\partial X_{k-1}}$. Let's assume again that we already know $\frac{\partial E}{\partial X_k}$, in other words, for each component of vector X_k we know how much E would wiggle if we wiggled that component of X_k.



- We can apply chain rule to compute $\frac{\partial E}{\partial X_{k-1}}$ (how much E would wiggle if we wiggled each component of X_{k-1}):

$$\frac{\partial E}{\partial X_{k-1}} = \frac{\partial E}{\partial X_k} \frac{\partial F_k(X_{k-1}, W_k)}{\partial X_{k-1}}$$

- $\frac{\partial F_k(X_{k-1}, W_k)}{\partial X_{k-1}}$ is the *Jacobian matrix* of F_k with respect to X_{k-1}.
- F_k has two Jacobian matrices, because it has to arguments.
- Element (p, q) of this Jacobian indicates how much the p-th output wiggles when we wiggle the q-th input.
- **The equation above is a recurrence equation!**

Jacobians and Dimensions

- derivatives with respect to a column vector are line vectors (dimensions: $[1 \times N_{k-1}] = [1 \times N_k] * [N_k \times N_{k-1}]$)

$$\frac{\partial E}{\partial X_{k-1}} = \frac{\partial E}{\partial X_k} \frac{\partial F_k(X_{k-1}, W_k)}{\partial X_{k-1}}$$

- (dimensions: $[1 \times N_{wk}] = [1 \times N_k] * [N_k \times N_{wk}]$):

$$\frac{\partial E}{\partial W_k} = \frac{\partial E}{\partial X_k} \frac{\partial F_k(X_{k-1}, W_k)}{\partial W}$$

- we may prefer to write those equation with column vectors:

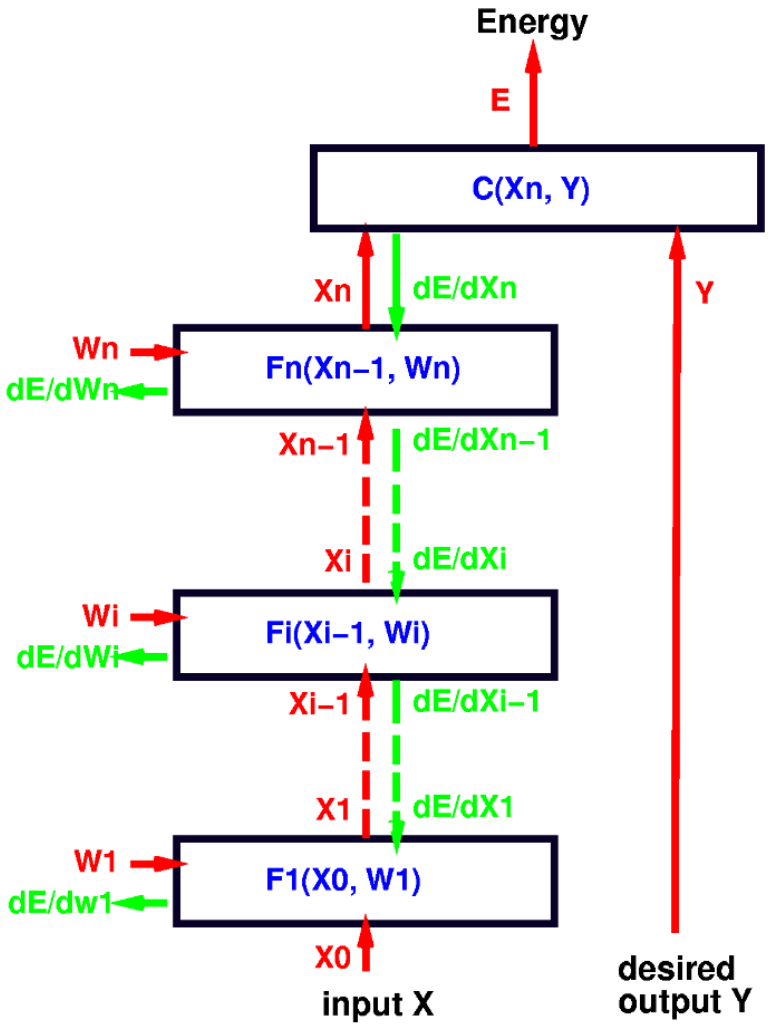
$$\frac{\partial E}{\partial X_{k-1}}' = \frac{\partial F_k(X_{k-1}, W_k)}{\partial X_{k-1}}' \frac{\partial E}{\partial X_k}'$$

$$\frac{\partial E}{\partial W_k}' = \frac{\partial F_k(X_{k-1}, W_k)}{\partial W}' \frac{\partial E}{\partial X_k}'$$

Back Propgation

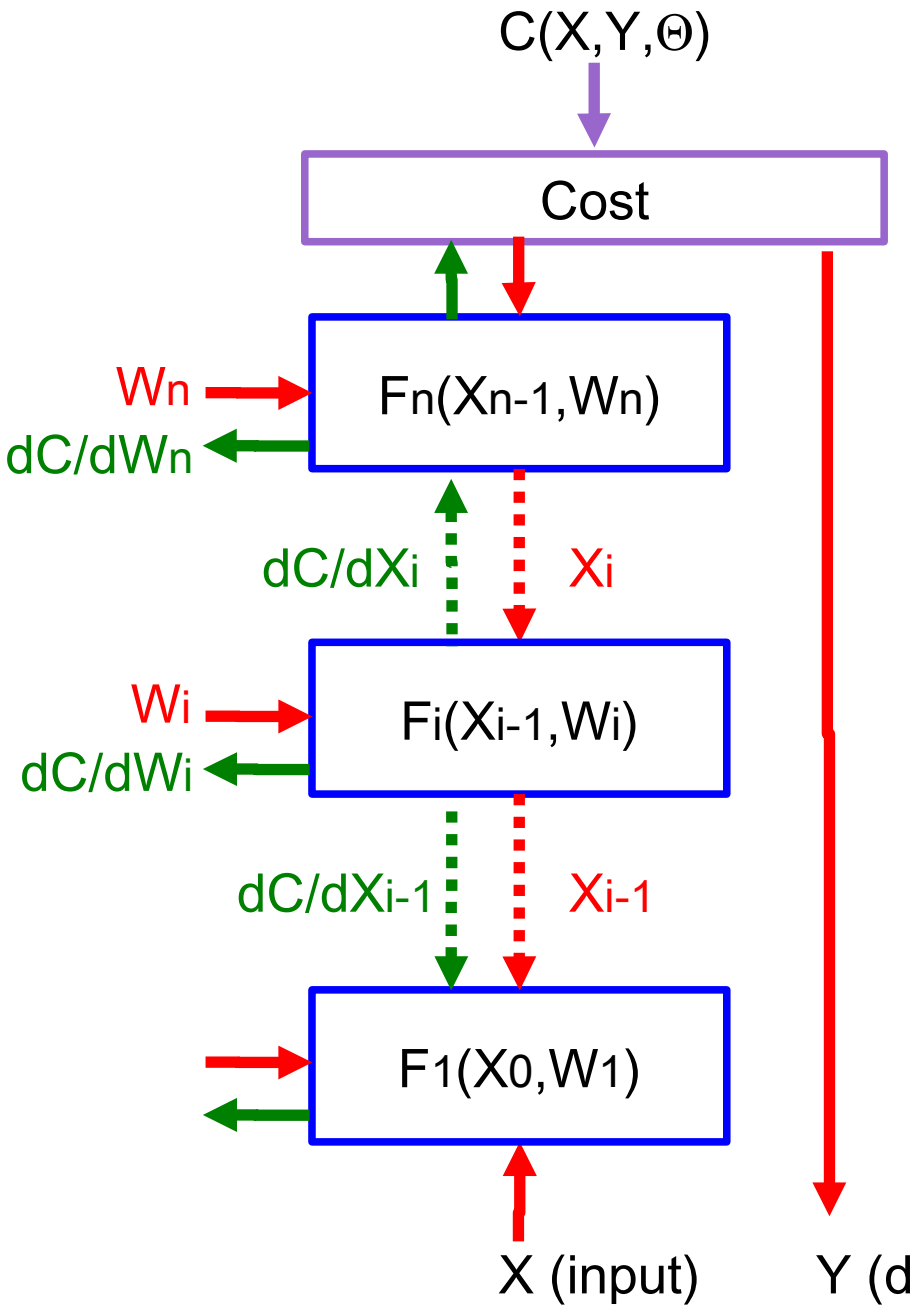
Y LeCun
MA Ranzato

To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for $\frac{\partial E}{\partial X_k}$



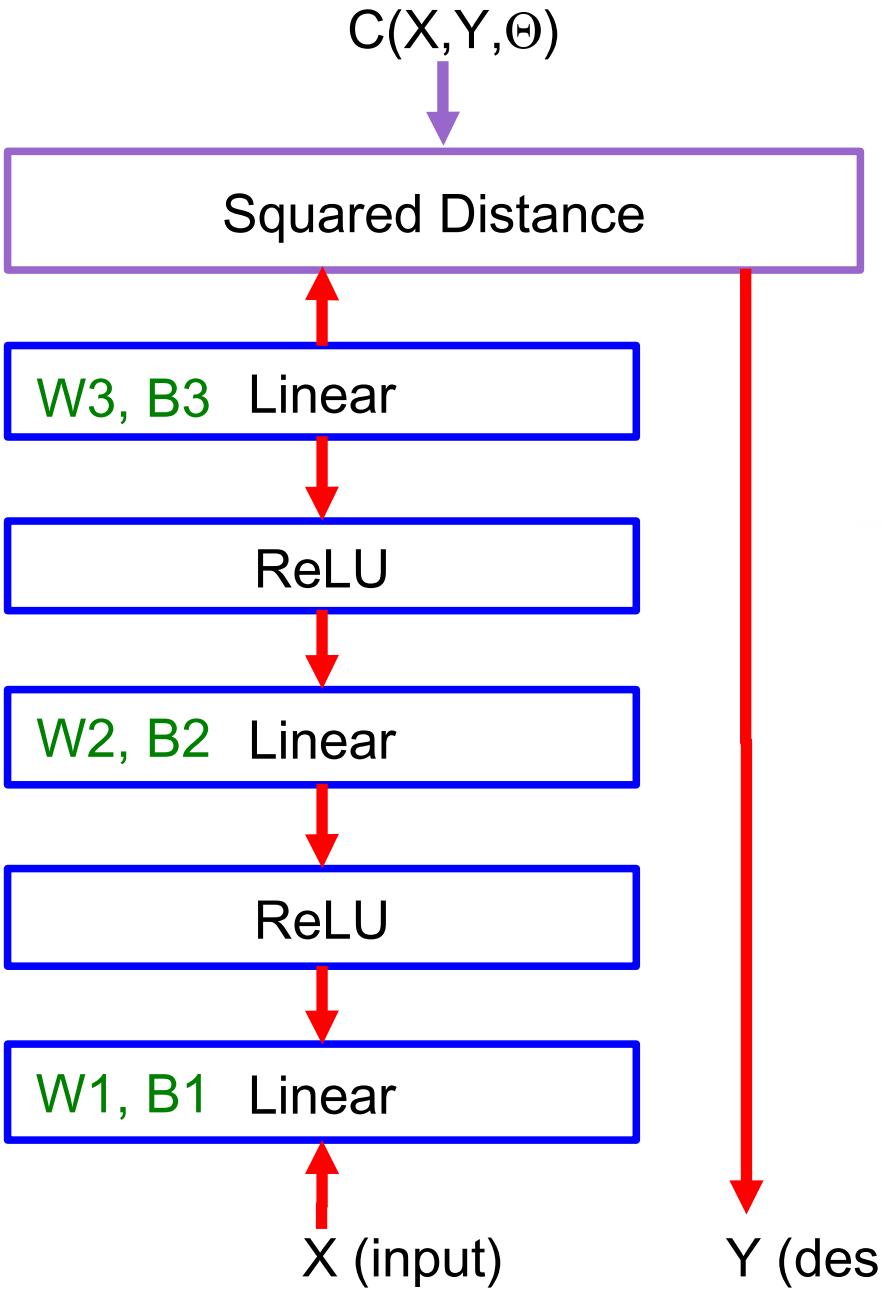
- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the $\frac{\partial E}{\partial W_k}$ for $k \in [1, n]$.

Computing Gradients by Back-Propagation



- **A practical Application of Chain Rule**
- **Backprop for the state gradients:**
- **dC/dXi-1 = dC/dXi . dXi/dXi-1**
- **dC/dXi-1 = dC/dXi . dFi(Xi-1,Wi)/dXi-1**
- **Backprop for the weight gradients:**
- **dC/dWi = dC/dXi . dXi/dWi**
- **dC/dWi = dC/dXi . dFi(Xi-1,Wi)/dWi**

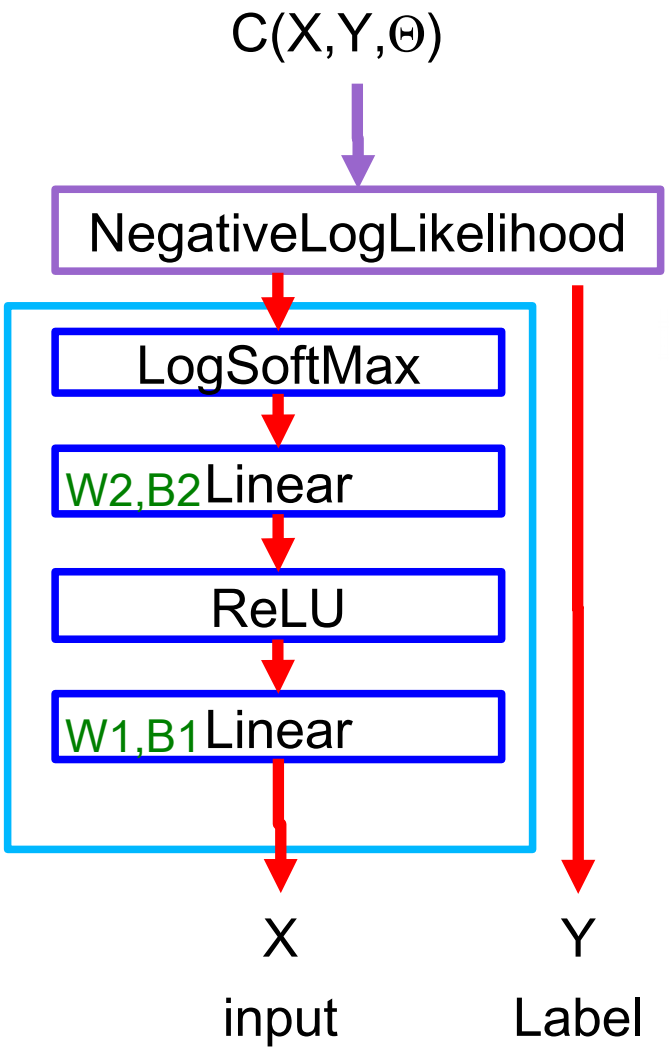
Typical Multilayer Neural Net Architecture



- **Complex learning machines can be built by assembling modules into networks**
- **Linear Module**
  - Out = W.In+B
- **ReLU Module (Rectified Linear Unit)**
  - $Out_i = 0$ if $In_i < 0$
  - $Out_i = In_i$ otherwise
- **Cost Module: Squared Distance**
  - $C = ||In1 - In2||^2$
- **Objective Function**
  - $L(\Theta) = 1/p \sum_k C(X^k, Y^k, \Theta)$
  - $\Theta = (W1, B1, W2, B2, W3, B3)$

Building a Network by Assembling Modules

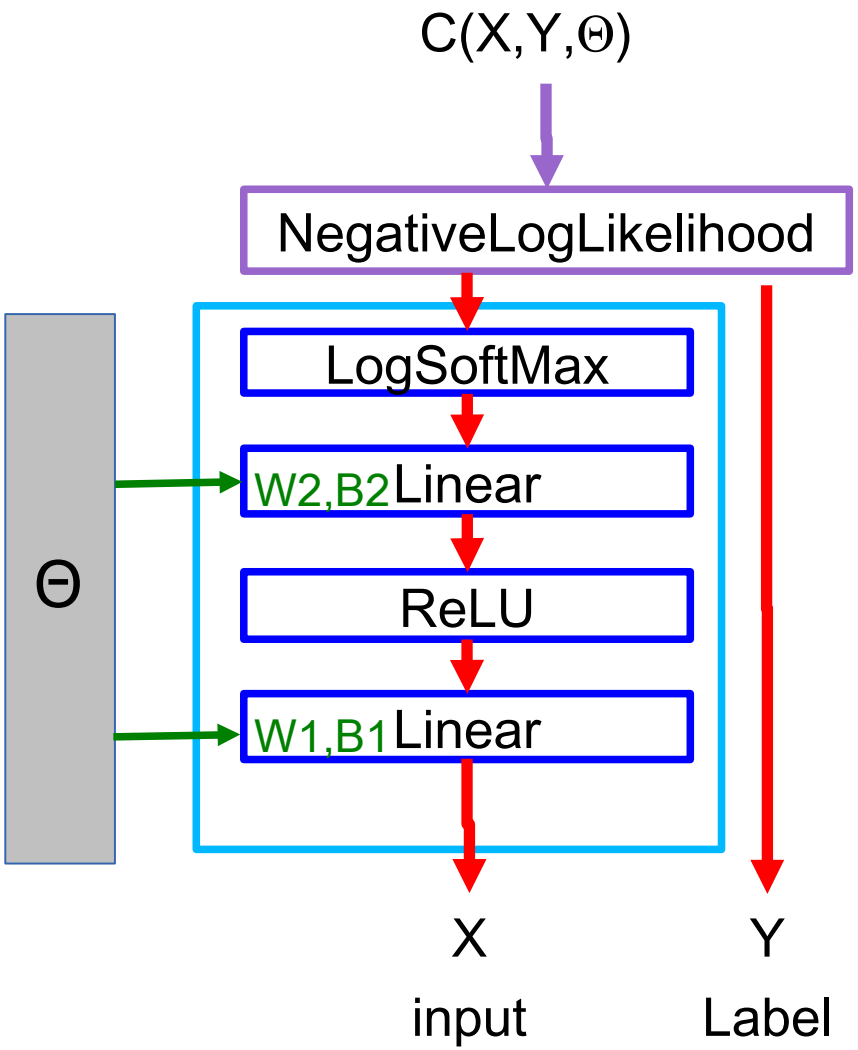
- All major deep learning frameworks use modules (inspired by SN/Lush, 1991)
  - Torch7, Theano, TensorFlow….




```
-- sizes
ninput = 28*28  -- e.g. for MNIST
nhidden1 = 1000
noutput = 10

-- network module
net = nn.Sequential()
net:add(nn.Linear(ninput, nhidden))
net:add(nn.Threshold())
net:add(nn.Linear(nhidden, noutput))
net:add(nn.LogSoftMax()))

-- cost module
cost = nn.ClassNLLCriterion()

-- get a training sample
input = trainingset.data[k]
target = trainingset.labels[k]

-- run through the model
output = net:forward(input)
c = cost:forward(output, target)
```

Running Backprop

- Torch7 example
- Gradtheta contains the gradient




```
-- network module
net = nn.Sequential()
net:add(nn.Linear(ninput, nhidden))
net:add(nn.Threshold())
net:add(nn.Linear(nhidden, noutput))
net:add(nn.LogSoftMax()))

-- cost module
cost = nn.ClassNLLCriterion()

-- gather the parameters in a vector
theta, gradtheta = net:getParameters()

-- get a training sample
input = trainingset.data[k]
target = trainingset.labels[k]

-- run through the model
output = net:forward(input)
c = cost:forward(output, target)

-- run backprop
gradtheta:zero()
gradoutput = cost:backward(output, target)
net:backward(input, gradoutput)
```

Module Classes

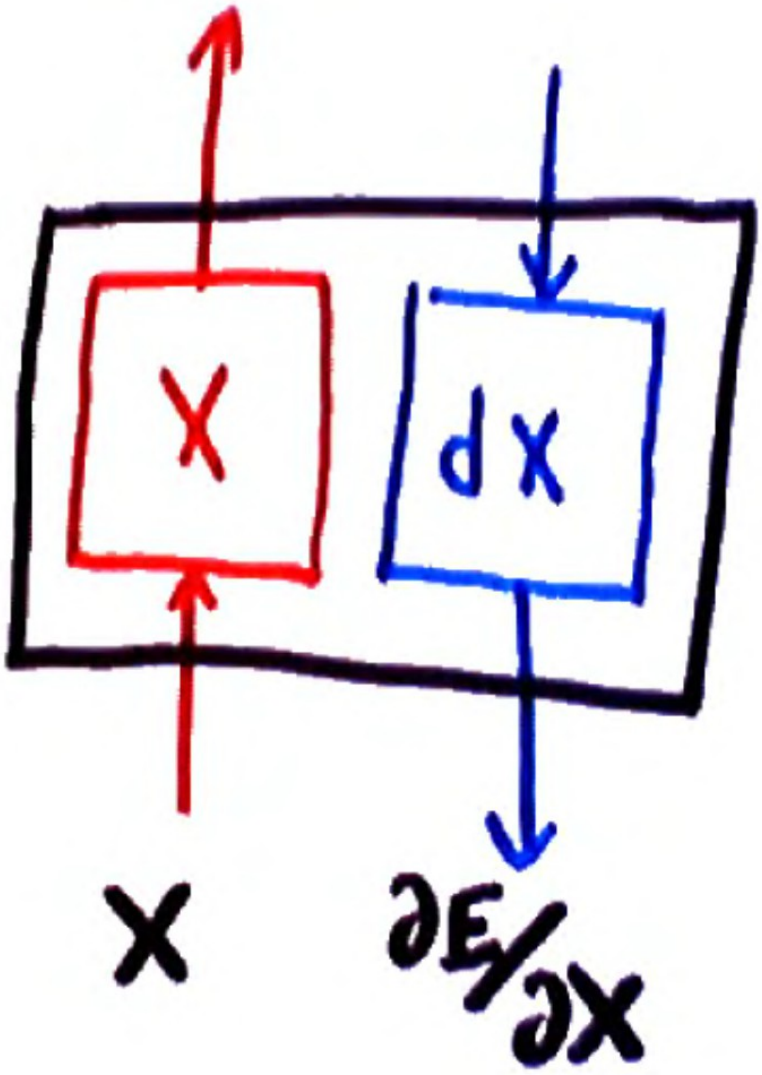
- **Linear** • $Y = W.X$; $dC/dX = W^T.\, dC/dY$; $dC/dW = dC/dY\, .\, X^T$
- **ReLU** • $y = ReLU(x)$; if $(x<0)$ $dC/dx = 0$ else $dC/dx = dC/dy$
- **Duplicate** • $Y1 = X,\ Y2 = X$; $dC/dX = dC/dY1 + dC/dY2$
- **Add** • $Y = X1 + X2$; $dC/dX1 = dC/dY$; $dC/dX2 = dC/dY$
- **Max** • $y = max(x1,x2)$; if $(x1>x2)$ $dC/dx1 = dC/dy$ else $dC/dx1=0$
- **LogSoftMax** • $Yi = Xi - \log[\sum_j \exp(Xj)]$; .....

Module Classes

- **Linear** • $Y = W.X$; $dC/dX = W^T.\, dC/dY$; $dC/dW = dC/dY\, .\, X^T$
- **ReLU** • $y = ReLU(x)$; if $(x<0)$ $dC/dx = 0$ else $dC/dx = dC/dy$
- **Duplicate** • $Y1 = X,\ Y2 = X$; $dC/dX = dC/dY1 + dC/dY2$
- **Add** • $Y = X1 + X2$; $dC/dX1 = dC/dY$; $dC/dX2 = dC/dY$
- **Max** • $y = max(x1,x2)$; if $(x1>x2)$ $dC/dx1 = dC/dy$ else $dC/dx1=0$
- **LogSoftMax** • $Yi = Xi - \log[\sum_j \exp(Xj)]$; .....

State Variables

Y LeCun
MA Ranzato

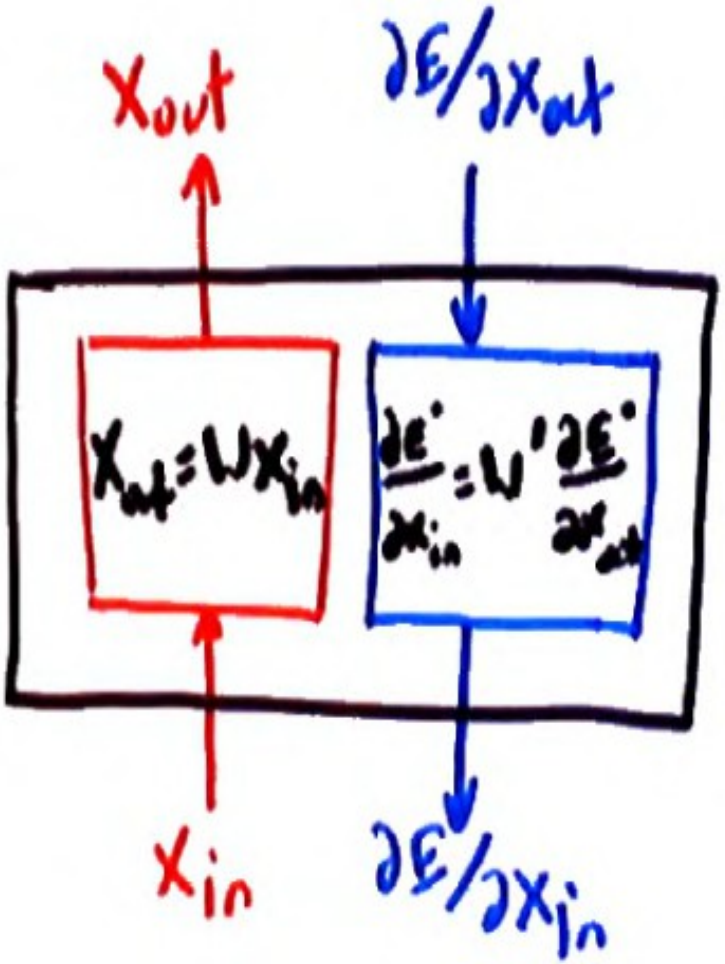


the internal state of the network will be kept in a "state" class that contains two scalars, vectors, or matrices: (1) the state proper, (2) the derivative of the energy with respect to that state.

Linear Module

Y LeCun
MA Ranzato

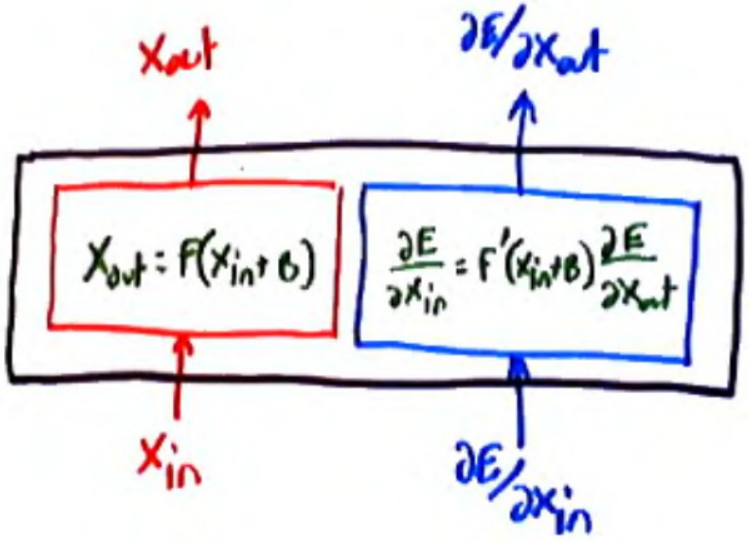
The input vector is multiplied by the weight matrix.



- fprop: $X_{\text{out}} = W X_{\text{in}}$
- bprop to input:
$$\frac{\partial E}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} \frac{\partial X_{\text{out}}}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} W$$
- by transposing, we get column vectors:
$$\frac{\partial E}{\partial X_{\text{in}}}' = W' \frac{\partial E}{\partial X_{\text{out}}}'$$
- bprop to weights:
$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial X_{\text{out}i}} \frac{\partial X_{\text{out}i}}{\partial W_{ij}} = X_{\text{in}j} \frac{\partial E}{\partial X_{\text{out}i}}$$
- We can write this as an outer-product:
$$\frac{\partial E}{\partial W}' = \frac{\partial E}{\partial X_{\text{out}}}' X_{in}'$$

Tanh module (or any other pointwise function)

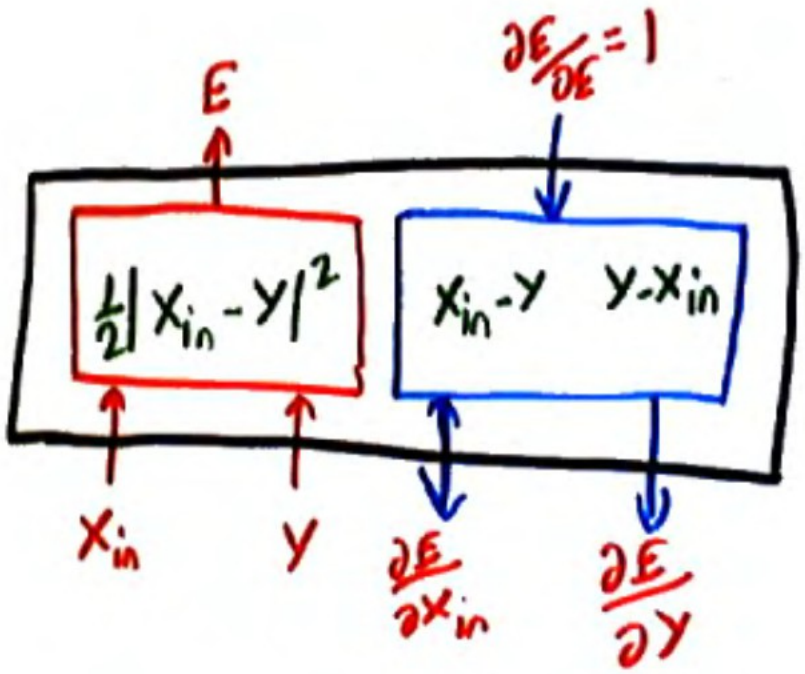
Y LeCun
MA Ranzato



- fprop: $(X_{\text{out}})_i = \tanh((X_{\text{in}})_i + B_i)$
- bprop to input:
  $(\frac{\partial E}{\partial X_{\text{in}}})_i = (\frac{\partial E}{\partial X_{\text{out}}})_i \tanh'((X_{\text{in}})_i + B_i)$
- bprop to bias:
  $\frac{\partial E}{\partial B_i} = (\frac{\partial E}{\partial X_{\text{out}}})_i \tanh'((X_{\text{in}})_i + B_i)$
- $\tanh(x) = \frac{2}{1+\exp -x} - 1 = \frac{1-\exp(-x)}{1+\exp(-x)}$

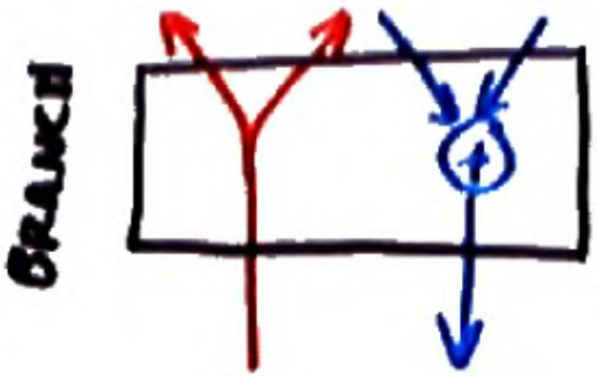
Euclidean Distance Module (Squared Error)

Y LeCun
MA Ranzato



- fprop: $X_{\text{out}} = \frac{1}{2}||X_{\text{in}} - Y||^2$
- bprop to X input: $\frac{\partial E}{\partial X_{\text{in}}} = X_{\text{in}} - Y$
- bprop to Y input: $\frac{\partial E}{\partial Y} = Y - X_{\text{in}}$

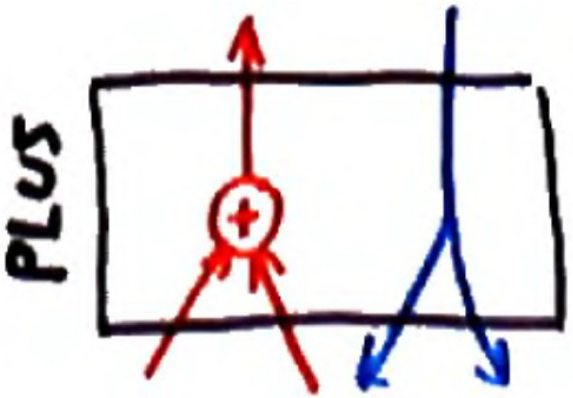
Y connector and Addition modules



- The PLUS module: a module with K inputs $X_1, \ldots, X_K$ (of any type) that computes the sum of its inputs:

$$X_{\text{out}} = \sum_k X_k$$

back-prop: $\frac{\partial E}{\partial X_k} = \frac{\partial E}{\partial X_{\text{out}}} \quad \forall k$

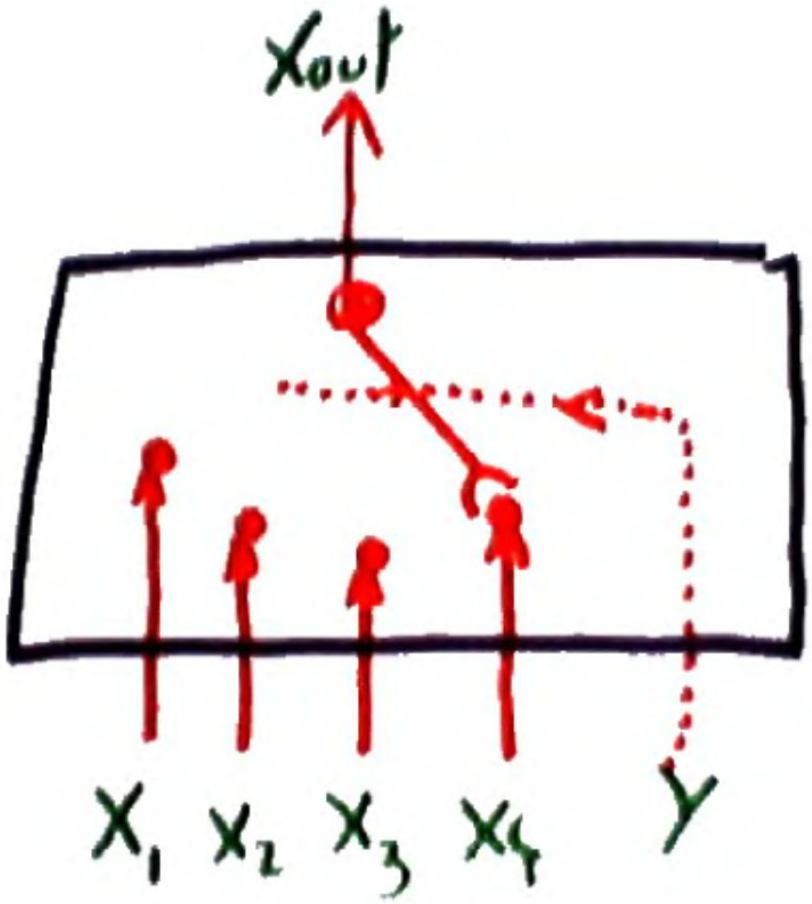


- The BRANCH module: a module with one input and K outputs $X_1, \ldots, X_K$ (of any type) that simply copies its input on its outputs:

$$X_k = X_{\text{in}} \quad \forall k \in [1..K]$$

back-prop: $\frac{\partial E}{\partial \text{in}} = \sum_k \frac{\partial E}{\partial X_k}$

Switch Module



- A module with K inputs $X_1, \ldots, X_K$ (of any type) and one additional discrete-valued input Y.
- The value of the discrete input determines which of the N inputs is copied to the output.
- $$X_{\text{out}} = \sum_k \delta(Y-k) X_k$$

  $$\frac{\partial E}{\partial X_k} = \delta(Y-k) \frac{\partial E}{\partial X_{\text{out}}}$$

  the gradient with respect to the output is copied to the gradient with respect to the switched-in input. The gradients of all other inputs are zero.

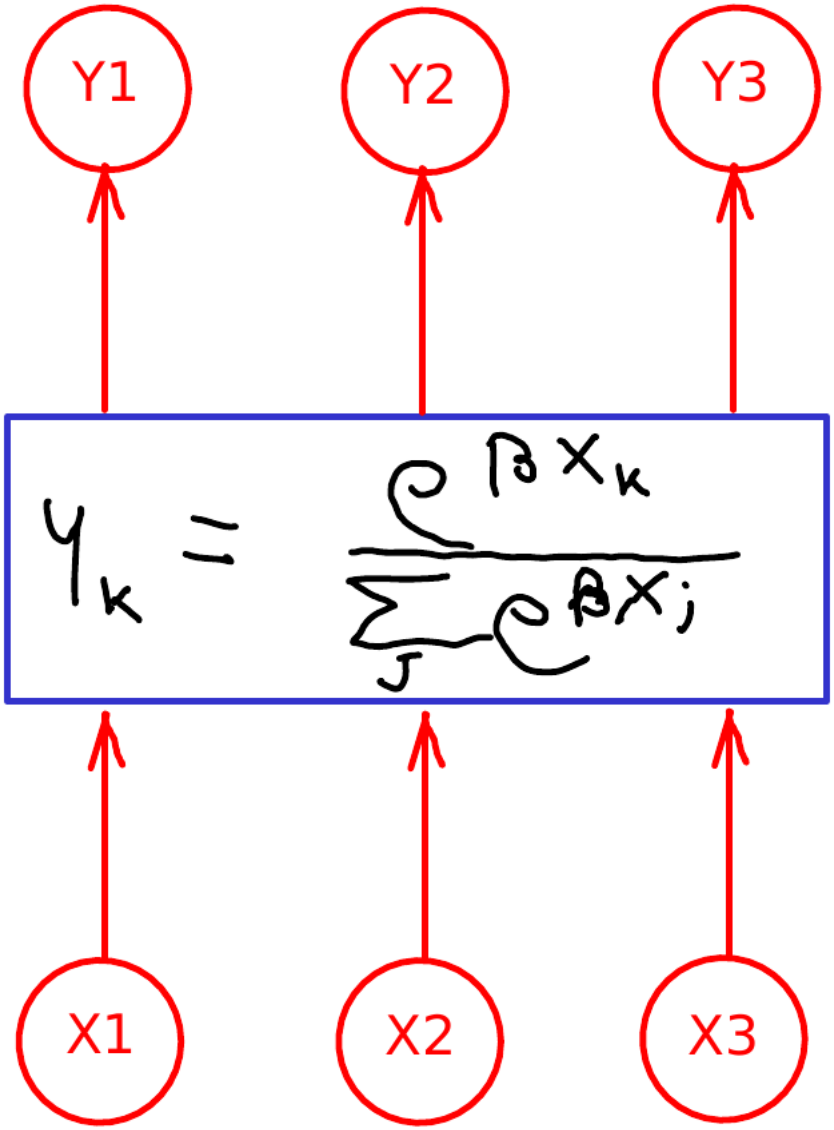
SoftMax Module (should really be called SoftArgMax)

Y LeCun
MA Ranzato

- Transforms scores into a discrete probability distribution
  - Positive numbers that sum to one.

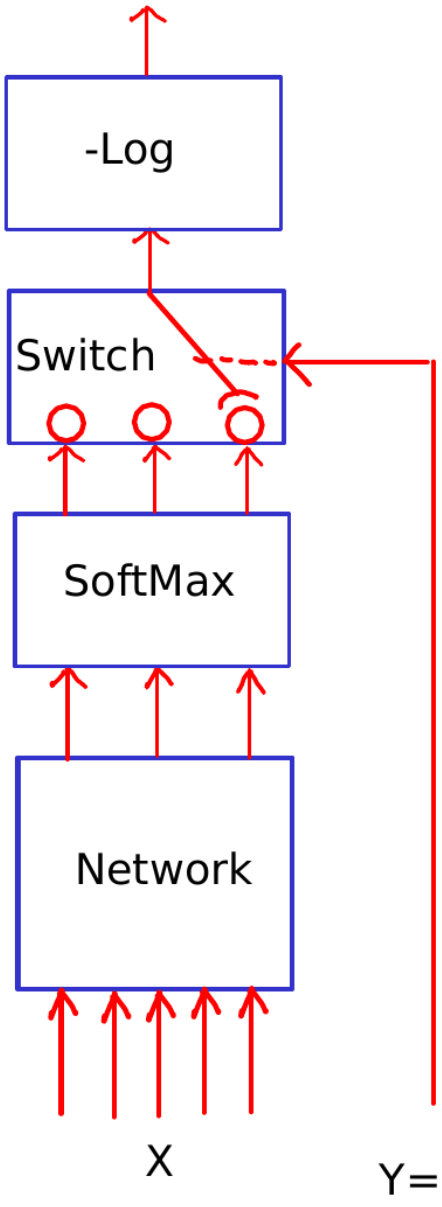
- Used in multi-class classification

$$P_k = \frac{e^{\beta X_k}}{\sum_j e^{\beta X_j}}$$

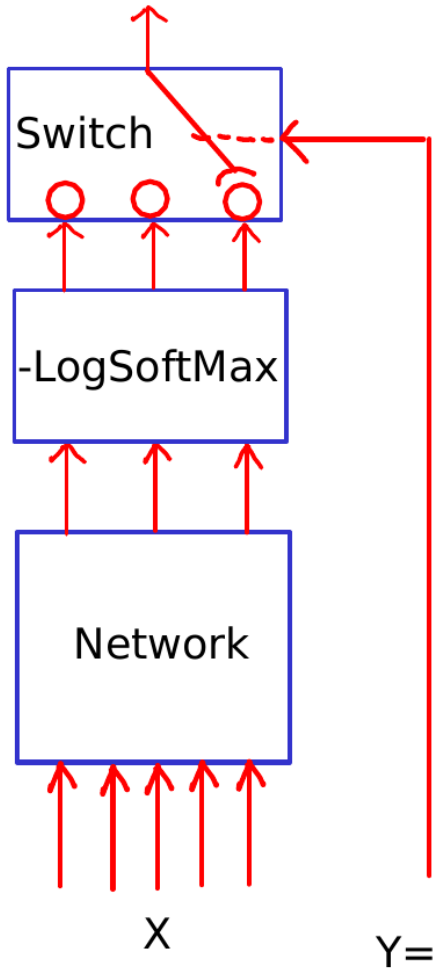


SoftMax Module: Loss Function for Classification

Y LeCun
MA Ranzato

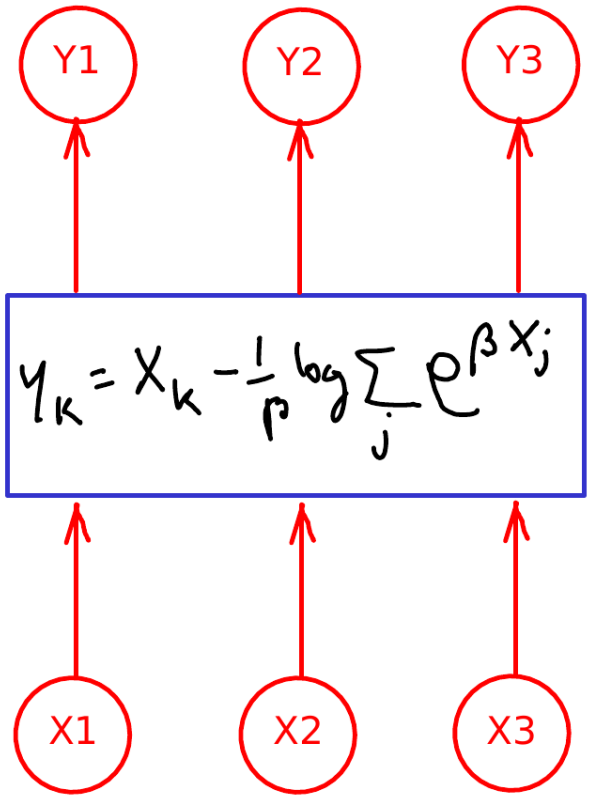


- -LogSoftMax: $-\frac{1}{\beta}\log p_k = -x_k + \frac{1}{\beta}\log\sum_j e^{\beta x_j}$
- Maximum conditional likelihood
- Minimize -log of the probability of the correct class.

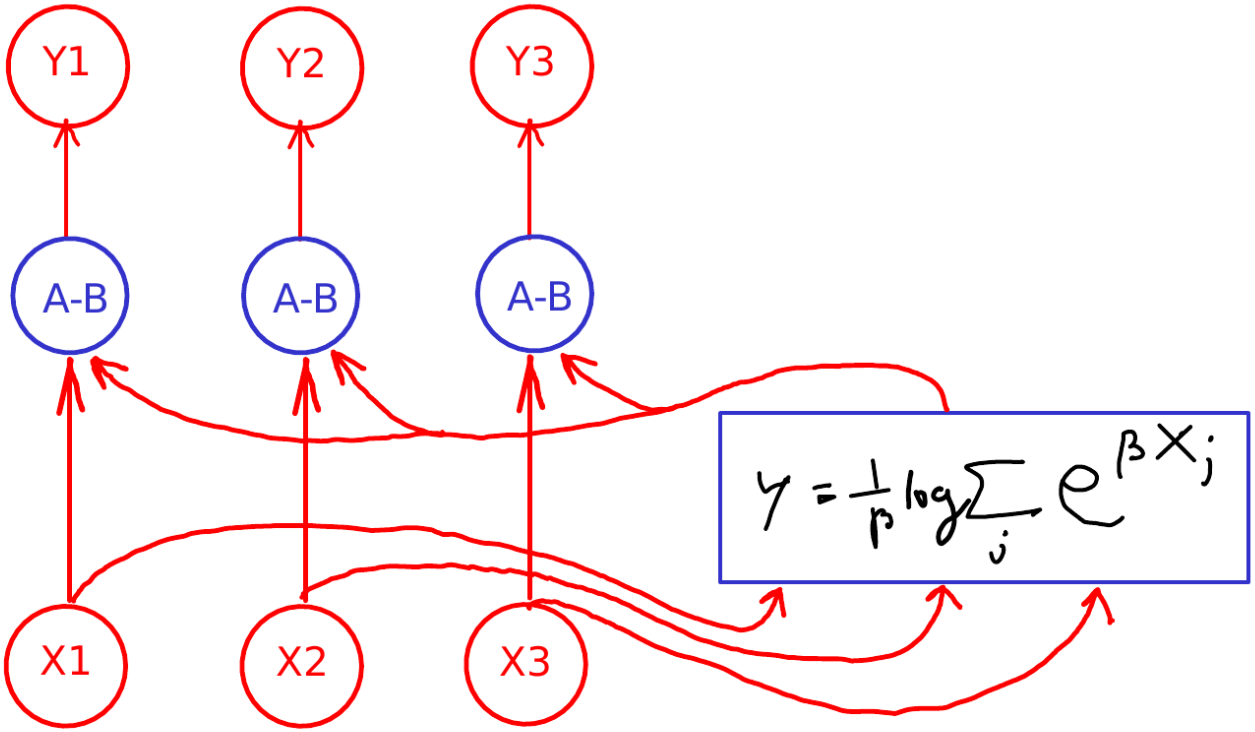


LogSoftMax Module

Y LeCun
MA Ranzato



- Transforms scores into a discrete probability distribution
- LogSoftMax = Identity - LogSumExp



LogSumExp Module

- Log of normalization term for SoftMax
- Fprop

$$X_{out} = \frac{1}{\beta} \sum_j e^{\beta X_j}$$

- Bprop

$$\frac{\partial E}{\partial X_k} = \frac{\partial E}{\partial X_{out}} \cdot \frac{e^{\beta X_k}}{\sum_j e^{\beta X_j}}$$

- Or:

$$\frac{\partial E}{\partial X_k} = \frac{\partial E}{\partial X_{out}} \cdot P_k \qquad P_k = \frac{e^{\beta X_k}}{\sum_j e^{\beta X_j}}$$